Learning Statistical Models Through Simulation in R

Dale J. Barr

2020-12-31

Contents

Overview

This textbook approaches statistical analysis through the General Linear Model, taking a simulation-based approach in the R software environment. The overarching goal is to teach students how to translate a description of the design of a study into a linear model to analyze data from that study. The focus is on the skills needed to analyze data from psychology experiments.

The following topics are covered:

- linear modeling workflow;
- variance-covariance matrices;
- multiple regression;
- interactions (continuous-by-categorical; categorical-by-categorical);
- linear mixed-effects regression;
- generalized linear mixed-effects regression.

The material in this course forms the basis for a one-semester course for third-year undergradautes taught by Dale Barr at the University of Glasgow School of Psychology. It is part of the PsyTeachR series of course materials developed by University of Glasgow Psychology staff.

Unlike other textbooks you may have encountered, this is an **interactive textbook**. Each chapter contains embedded exercises as well as web applications to help students better understand the content. The interactive content will only work if you access this material through a web browser. Printing out the material is not recommended. If you want to access the textbook without an internet connection or have a local version to keep in case this site changes or moves, you can download a version for offline use. Just extract the files from the ZIP archive, locate the file `index.html` in the `docs` directory, and open this file using a web browser.

How to cite this book

Barr, Dale J. (2020). *Learning statistical models through simulation in R: An interactive textbook*. Version 0.9-3. Retrieved from https://psyteachr.github.io/book/ug3-stats.

Information for educators

The book was built using the R **`bookdown`** package. The source files are available at github.

Chapter 1

Introduction

1.1 Goal of this course

The goal of this course is to teach you how to simulate and analyze the various types of datasets you are likely to encounter as a psychologist. The focus is on behavioral data—e.g., measures of reaction time, perceptual judgments, choices, decision, ratings on a survey, hours of sleep, etc.—usually collected in the context of a study or experiment.

This course emphasizing approaches that maximize **flexibility**, **generalizability**, and **reproducibility**. You may have heard talk in the media about psychology being in the throes of various 'crises'. There have been some high-profile replication failures that have contributed to the impression that all is not well. There have also been a number of demonstrations that the standard approaches to data analysis in psychology are flawed, in that they allow researchers to present results as clean-cut and confirmatory while hiding various analyses that have been pursued and discarded while seeking to attain the desired result. But it is not just the field of psychology where such problems can be found—the problem is widespread, encompassing other fields of study as well. And the truth is that a certain level of self-criticism and self-reflection can also be seen positively—as a sign of a healthy discipline. Whatever your take, there is room for improvement, and the purpose of this course is to give you tools to ensure that your own analyses are not only statistically sound, but also as generalizable and reproducible as possible.

1.1.1 Flexibility

The approach taken in this course differs from other courses in that it emphasizes learning a flexible regression modeling framework rather than teaching 'recipes' for dealing with different types of data, and assumes that the fundamental type of data you will need to analyze will be multilevel and also that you will need to deal not only with continuous measurements but also ordinal measurements (ratings on a Likert scale), counts (number of times particular event types have taken place), and nominal data (what category something falls into).

The most flexible approach available is the General Linear Model approach, in which some response variable is modeled in terms of a weighted linear sum of predictor variables. A simple mathematical example of one such formula is

$$Y_i = \beta_0 + \beta_1 X_i + e_i$$

where Y_i is the measured response for observation i, modeled in terms of an intercept term plus the effect of predictor X_i weighted by coefficient β_1 and residual error term e_i, which reflects unmodeled variance.

You might recognize the above equation as representing the equation for a line ($y = mx + b$), with β_0 playing the role of the y-intercept and β_1 playing the role of the slope. The e_i term reflects what is left of observation i after accounting for the the intercept and the predictor X_i.

A terminological convention in this course is that Greek letters (β, ρ, τ) represent unobserved population parameters while Latin letters (X, Y) represent observed values.

This linear formulation is highly general, and we will see how it can be used to model different kinds of relationships between different kinds of variables. One limitation of the current course is that it focuses mainly on univariate data, where a single response variable is the focus of analysis. It is often the case that you will be dealing with multiple response variables on the same subjects, but modeling them all simultaneously is technically very difficult and beyond the scope of this course. A simpler approach (and the one adopted here) is to analyze each response variable in a separate univariate analysis.

1.2 Generalizability

The term generalizability refers to the degree to which findings from a study can be readily applied to situations beyond the particular context (subjects, stimuli, task, etc.) in which the data were collected. At best, our findings would apply to all members of the human species across a wide variety of contexts; at worst, they would apply only to those specific people observed in the specific context of our study. Most studies fall somewhere between these two opposing poles.

The generalizability of a finding depends on several factors: how the study was designed, what materials were used, how subjects were recruited, the nature of the task given to the subjects, **and the way the data were analyzed**. It is this last point that we will focus on in this course. When analyzing a dataset, if you want to make generalizable claims, you must make decisions about what would count as a replication of your study—about which aspects would remain fixed across replications, and which aspects you would allow to vary. Unfortunately, you will sometimes find that data have been analyzed in a way that does not support generalization in the broadest sense, often because analyses underestimate the influence of ideosyncratic features of stimulus materials or experimental task on the observed result (Yarkoni, 2019).

1.3 Reproduciblity and Transparency

The term reproducibility refers to the degree to which it is possible to reproduce the pattern of findings in a study under various circumstances.

We say a finding is *analytically* or *computationally* reproducible if, given the raw data, we can obtain the same pattern of results. Note that this is different from saying a finding is **replicable**, which refers to being able to reproduce the finding in **new samples.** There is not widespread agreement about these terms, but it is convenient to view analytic reproducibility and replicability as two different but related types of reproducibility, with the former capturing reproducibility across analysts (or among the same analysts over time) and the latter reflecting reproducibility across participants.

Ensuring that analyses are reproducible is a hard problem. If you fail to properly document your own analyses, or the software that you used gets modified or goes out of date and becomes unavailable, you may have trouble reproducing your own findings!

Another important property of analyses is transparency: the extent to which all the steps in some research study have been made available. A study may be transparent but not reproducible, and vice versa. It is important to use a workflow that promotes transparency. This makes the 'edit-and-execute' workflow of script programming ideal for data analysis, far superior to the 'point-and-click' workflow of most commercial statistical programs. By writing code, you make the logic and decision process explicit to others and easy to reconstruct.

1.4 A simulation-based approach

A final characteristic that distinguishes this course is that it uses a **simulation-based approach** to learning about statistical models. By data simulation we mean specifying a model to characterize a population of interest and then using the computer's random number generator to simulate the process of sampling data from that population. We will look at a simple example of this below.

The typical problem that you will face when you analyze data is that you won't know the 'ground truth' about the population you are studying. You take a sample from that population, make some assumptions about how the observed data have been generated, and then use the observed data to estimate unknown population parameters and the uncertainty around these parameters.

Data simulation inverts this process. You to define the parameters of a model representing the ground truth about a (hypothetical) population and then generate data from it. You can then analyze the resulting data in the way that you normally would, and see how well your parameter estimates correspond to the true values.

Let's look at an example. Let's assume you are interested in the question of whether being a parent of a toddler 'sharpens' your reflexes. If you've ever taken care of a toddler, you know that physical danger always seems imminent—they could fall of the chair they just climbed on, slam their finger in a door, bang their head on the corner of a table, etc.—so you need to be attentive and ready to act fast. You hypothesize that this vigilance will translate into faster response times in other situations where a toddler is not around, such as in a psychological laboratory. So you recruit a set of parents of toddlers to come into the lab. You give each parent the task of pressing a button as quickly as possible in response to a flashing light, and measure their response time (in milliseconds). For each parent, you calculate their mean response time over all trials. We can simulate the mean response time for each of say, 50 parents using the `rnorm()` function in R. But before we do that, we will load in the packages that we need (tidyverse) and set the random seed to make sure that you (the reader) get the same random values as me (the author).

```r
library("tidyverse")

set.seed(2020)  # can be any arbitrary integer
```

```r
parents <- rnorm(n = 50, mean = 480, sd = 40)

parents
```

```
##  [1] 495.0789 492.0619 436.0791 434.7838 368.1386 508.8229 517.5648 470.8249
##  [9] 550.3653 484.6947 445.8751 516.3704 527.8549 465.1366 475.0696 552.0017
## [17] 548.1598 358.4494 388.4410 482.3321 566.9746 523.9273 492.7288 477.0741
## [25] 513.3707 487.9500 531.9137 517.4687 474.1027 484.4173 447.4998 450.2519
## [33] 523.8138 577.4149 495.5247 491.6251 468.5761 483.0406 457.5881 497.8875
## [41] 516.3400 459.7976 467.9598 450.9586 432.7969 490.1230 465.1715 480.8872
## [49] 506.4018 499.5517
```

We chose to generate the data using `rnorm()`—a function that generates random numbers from a normal distribution—reflecting our assumption that mean response time is normally distributed in the population. A normal distribution is defined by two parameters, a mean (usually notated with the Greek letter μ, pronounced "myoo"), and a standard deviation (usually notated with the Greek letter σ, pronounced "sigma"). Since we have generated the data ourselves, both μ and σ are known, and in the call to `rnorm()`, we set them to the values 480 and 40 respectively.

But of course, to test our hypothesis, we need a comparison group, so we define a control group of non-parents. We generate data from this control group in the same way as above, but changing the mean.

```
control <- rnorm(n = 50, mean = 500, sd = 40)
```

Let's put them into a tibble to make it easier to plot and analyze the data. Each row from this table represents the mean response time from a particular subject.

```
dat <- tibble(group = rep(c("parent", "control"), c(length(parents), length(control))),
              rt = c(parents, control))

dat
```

```
## # A tibble: 100 x 2
##    group     rt
##    <chr>  <dbl>
##  1 parent  495.
##  2 parent  492.
##  3 parent  436.
##  4 parent  435.
##  5 parent  368.
##  6 parent  509.
##  7 parent  518.
##  8 parent  471.
##  9 parent  550.
## 10 parent  485.
## # ... with 90 more rows
```

Here are some things to try with this simulated data.

1. Plot the data in some sensible way.
2. Calculate means and standard deviations. How do they compare with the population parameters?
3. Run a t-test on these data. Is the effect of group significant?

Once you have done these things, do them again, but changing the sample size, population parameters or both.

1.5 What you will learn

By the end of this course, you should be able to:

- simulate bivariate data
- understand the relationship between correlation and regression
- specify regression models to reflect various study designs
- specify and interpret various types of interactions
- simulate data from these models, including multilevel data
- handle continuous, count, or nominal dependent variables.

Chapter 2

Understanding correlation and regression through bivariate simulation

- Packages used:
  - `tidyverse`
  - `corrr`
  - `MASS` (built in)
- Functions used:
  - `base::choose()`
  - `corrr::correlate()`
  - `base::matrix()`
  - `base::cor()`
  - `base::log()` and `base::exp()` for log transformation (and back)
- Review
  - L2 lab materials on correlation.

2.1 Correlation matrices

You may be familiar with the concept of a **correlation matrix** from reading papers in psychology. Correlation matrices are a common way of summarizing relationships between multiple measurements taken from the same individual.

Let's say you've measured psychological well-being using multiple scales. One question is the extent to which these scales are measuring the same thing. Often you will look at a correlation matrix to explore all the pairwise relationships between measures.

Recall that a correlation coefficient quantifies the **strength** and **direction** of a relationship between two variables. It is usually represented by the symbol r (for a **statistic**) and ρ (Greek letter "rho") (as a **parameter**). The coefficient ranges between -1 and 1, with 0 corresponding to no relationship, positive values reflecting a positive relationship (as one variable increases, so does the other), and negative values reflecting a negative relationship (as one variable increases, the other decreases).

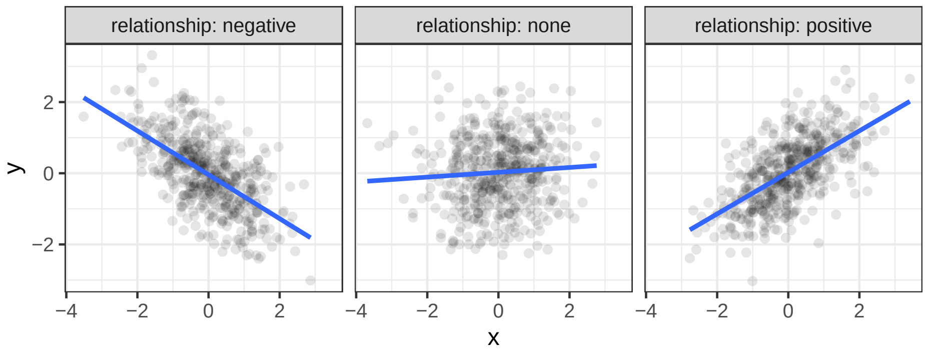

Figure 2.1: Different types of bivariate relationships.

If you have n measures, how many pairwise correlations can you compute? You can figure this out either by the formula in the info box below, or more easily you can computed it directly through the `choose(n, 2)` function in R. For instance, to get the number of possible pairwise correlations between 6 measures, you'd type `choose(6, 2)`, which tells you that you have 15 combinations.

For any n measures, you can calculate $\frac{n!}{2(n-2)!}$ pairwise correlations between measures. The ! symbol is called the **factorial** operator, defined as the product of all numbers from 1 to n. So, if you have six measurements, you have

$$\frac{6!}{2(6-2)!} = \frac{1 \times 2 \times 3 \times 4 \times 5 \times 6}{2\left(1 \times 2 \times 3 \times 4\right)} = \frac{720}{2(24)} = 15$$

You can create a correlation matrix in R using `base::cor()` or `corrr::correlate()`. We prefer the latter function because `cor()` requires that your data is stored in a matrix, whereas most of the data we will be working with is stored in a data frame. The `corrr::correlate()` function also takes a data frame as the first argument, so it works better with pipes (`%>%`).

Let's create a correlation matrix to see how it works. Start by loading in the packages we will need.

```r
library("tidyverse")
library("corrr")  # install.packages("corrr") in console if missing
```

We will use the `starwars` dataset, which is a built-in dataset that becomes available after you load the tidyverse package. This dataset has information about various characters that have appeared in the Star Wars film series. Let's look at the correlation between

```
starwars %>%
  select(height, mass, birth_year) %>%
  correlate()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'

## # A tibble: 3 x 4
##   rowname    height   mass birth_year
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 height     NA      0.134     -0.400
## 2 mass        0.134 NA          0.478
## 3 birth_year -0.400  0.478     NA
```

You can look up any bivariate correlation at the intersection of any given row or column. So the correlation between `height` and `mass` is .134, which you can find in row 1, column 2 or row 2, column 1; the values are the same. Note that there are only `choose(3, 2)` = 3 unique bivariate relationships, but each appears twice in the table. We might want to show only the unique pairs. We can do this by appending `corrr::shave()` to our pipeline.

```
starwars %>%
  select(height, mass, birth_year) %>%
  correlate() %>%
  shave()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'

## # A tibble: 3 x 4
##   rowname    height   mass birth_year
##   <chr>       <dbl>  <dbl>      <dbl>
## 1 height     NA     NA             NA
## 2 mass        0.134 NA             NA
## 3 birth_year -0.400  0.478         NA
```

Now we've only got the lower triangle of the correlation matrix, but the `NA` values are ugly and so are the leading zeroes. The **`corrr`** package also provides the `fashion()` function that cleans things up (see `?corrr::fashion` for more options).

```
starwars %>%
  select(height, mass, birth_year) %>%
  correlate() %>%
  shave() %>%
  fashion()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```
##      rowname height mass birth_year
## 1     height
## 2       mass    .13
## 3 birth_year   -.40  .48
```

Correlations only provide a good description of the relationship if the relationship is (roughly) linear and there aren't severe outliers that are wielding too strong of an influence on the results. So it is always a good idea to visualize the correlations as well as to quantify them. The `base::pairs()` function does this. The first argument to `pairs()` is simply of the form `~ v1 + v2 + v3 + ... + vn` where `v1`, `v2`, etc. are the names of the variables you want to correlate.

```
pairs(~ height + mass + birth_year, starwars)
```

We can see that there is a big outlier influencing our data; in particular, there is a creature with a mass greater than 1200kg! Let's find out who this is and eliminate them from the dataset.

```
starwars %>%
  filter(mass > 1200) %>%
  select(name, mass, height, birth_year)
```

```
## # A tibble: 1 x 4
##   name                   mass height birth_year
##   <chr>                 <dbl>  <int>      <dbl>
## 1 Jabba Desilijic Tiure  1358    175        600
```

OK, let's see how the data look without this massive creature.

```
starwars2 <- starwars %>%
  filter(name != "Jabba Desilijic Tiure")

pairs(~height + mass + birth_year, starwars2)
```

Better, but there's a creature with an outlying birth year that we might want to get rid of.

```
starwars2 %>%
  filter(birth_year > 800) %>%
  select(name, height, mass, birth_year)
```

```
## # A tibble: 1 x 4
##   name  height  mass birth_year
```

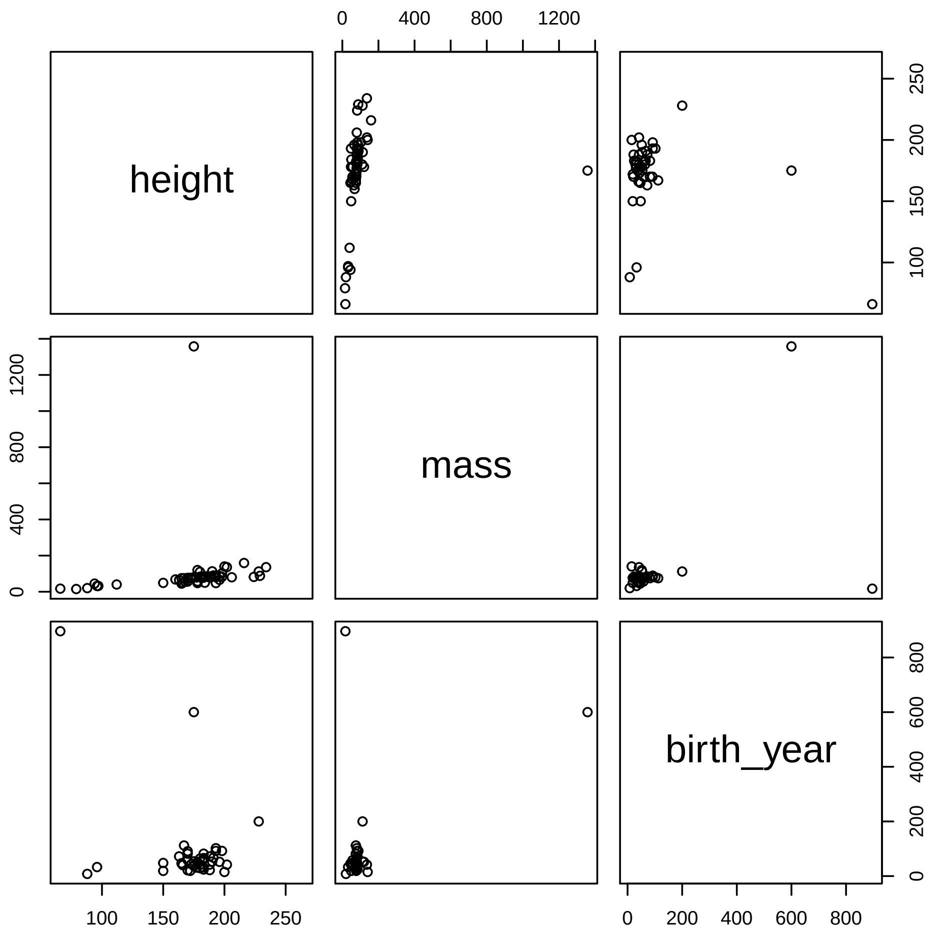

Figure 2.2: Pairwise correlations for the starwars dataset

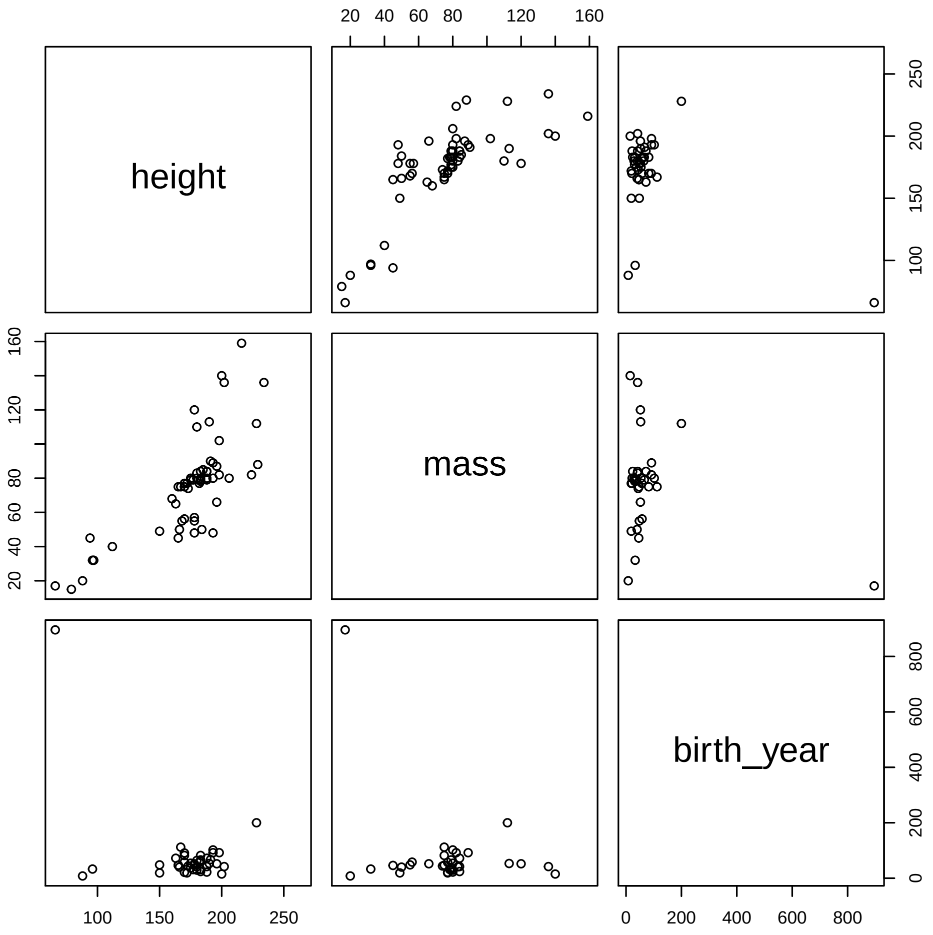

Figure 2.3: Pairwise correlations for the starwars dataset after removing outlying mass value.

```
##    <chr>  <int> <dbl>       <dbl>
## 1 Yoda       66    17         896
```

It's Yoda. He's as old as the universe. Let's drop him and see how the plots look.

```
starwars3 <- starwars2 %>%
  filter(name != "Yoda")

pairs(~height + mass + birth_year, starwars3)
```

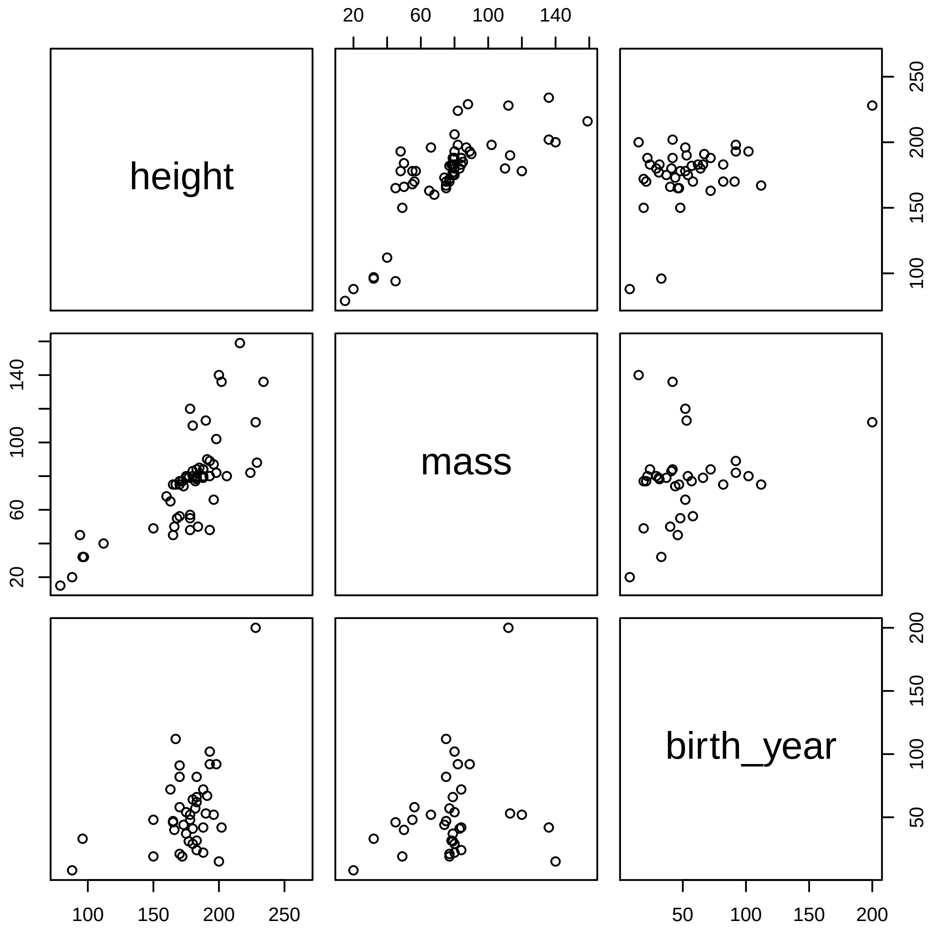

Figure 2.4: Pairwise correlations for the starwars dataset after removing outlying mass and birth_year values.

That looks much better. Let's see how that changes our correlation matrix.

```
starwars3 %>%
  select(height, mass, birth_year) %>%
  correlate() %>%
  shave() %>%
  fashion()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```
##      rowname height mass birth_year
## 1     height
## 2       mass    .74
## 3 birth_year    .45  .24
```

Note that these values are quite different from the ones we started with.

Sometimes it's not a great idea to remove outliers. Another approach to dealing with outliers is to use a robust method. The default correlation coefficient that is computed by `corrr::correlate()` is the Pearson product-moment correlation coefficient. You can also compute the Spearman correlation coefficient by changing the `method()` argument to `correlate()`. This replaces the values with ranks before computing the correlation, so that outliers will still be included, but will have dramatically less influence.

```
starwars %>%
  select(height, mass, birth_year) %>%
  correlate(method = "spearman") %>%
  shave() %>%
  fashion()
```

```
##
## Correlation method: 'spearman'
## Missing treated using: 'pairwise.complete.obs'
```

```
##      rowname height mass birth_year
## 1     height
## 2       mass    .75
## 3 birth_year    .16  .15
```

Incidentally, if you are generating a report from R Markdown and want your tables to be nicely formatted you can use `knitr::kable()`.

```
starwars %>%
  select(height, mass, birth_year) %>%
  correlate(method = "spearman") %>%
  shave() %>%
  fashion() %>%
  knitr::kable()
```

rowname	height	mass	birth_year
height			
mass	.75		
birth_year	.16	.15	

2.2 Simulating bivariate data

You have already learned how to simulate data from the normal distribution using the function `rnorm()`. Recall that `rnorm()` allows you to specify the mean and standard deviation of a single variable. How do we simulate correlated variables?

It should be clear that you can't just run `rnorm()` twice and combine the variables, because then you end up with two variables that are unrelated, i.e., with a correlation of zero.

The package **`MASS`** provides a function `mvrnorm()` which is the 'multivariate' version of rnorm (hence the function name, `mv` + `rnorm`, which makes it easy to remember.

The **`MASS`** package comes pre-installed with R. But the only function you'll probably ever want to use from **`MASS`** is `mvrnorm()`, so rather than load in the package using `library("MASS")`, it is preferable to use `MASS::mvrnorm()`, especially as **`MASS`** and the **`dplyr`** package from **`tidyverse`** don't play well together, due to both packages having the function `select()`. So if you load in **`MASS`** after you load in **`tidyverse`**, you'll end up getting the **`MASS`** version of `select()` instead of the **`dplyr`** version. It will do your head in trying to figure out what is wrong with your code, so always `MASS::mvrnorm` and never `library("MASS")`.

Have a look at the documentation for the `mvrnorm()` function (type `?MASS::mvrnorm` in the console).

There are three arguments to take note of:

arg	description
n	the number of samples required
mu	a vector giving the means of the variables
Sigma	a positive-definite symmetric matrix specifying the covariance matrix of the variables.

The descriptions for `n` and `mu` are understandable, but what is a "positive-definite symmetric matrix specifying the covariance"?

When you have multivariate data, the **covariance matrix** (also known as the **variance-covariance** matrix) specifies the variances of the individual variables and their interrelationships. It is like a multidimensional version of the **standard deviation**. To fully describe a univariate normal distribution, you need to know only the mean and standard deviation; to describe a bivariate normal distribution, you need the means of each of the two variables, their standard deviations, and their correlation; for a multivariate distribution with more the two variables you need the means for all of the variables, their standard deviations, and all of the possible pairwise correlations. **These concepts become very important once we start talking about mixed-effects modelling.**

You can think of a covariance matrix as something like the correlation matrix that you saw above; indeed, with a few calculations you can turn a covariance matrix into a correlation matrix.

What's all this talk about the Matrix? Wasn't that a sci-fi film series from the 1990s?

In mathematics, matrices are just generalizations of the concept of a vector: a vector can be thought of as having one dimension, whereas a matrix can have any number of dimensions.

So the matrix

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

is a 3 (row) by 3 (column) matrix containing the column vectors $\begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}$, $\begin{pmatrix} 4 \\ 5 \\ 6 \end{pmatrix}$, and $\begin{pmatrix} 7 \\ 8 \\ 9 \end{pmatrix}$. Conventionally, we refer to matrices in i by j format, with i being the number of rows and j being the number of columns. So a 3x2 matrix has 3 rows and 2 columns, like so.

$$\begin{pmatrix} a & d \\ b & e \\ c & f \end{pmatrix}$$

A **square matrix** is a matrix where the number of rows is equal to the number of columns.

You can create the above matrix in R using the `matrix()` function (see below) or by binding together vectors column-wise, using the base R `cbind()` function. Try `cbind(1:3, 4:6, 7:9)` in your console. You can also bind vectors together row-wise using `rbind()`.

Now what is all this about the matrix being "positive-definite" and "symmetric"? These are mathematical requirements about the kinds of matrices that can represent possible multivariate normal distributions. In other words, the covariance matrix you supply must be represent a legal multivariate normal distribution. At this point, you don't really need to know much more about this than that.

Let's start by simulating data representing hypothetical humans and their heights and weights. We know these things are correlated. What we need to be able to simulate data are means and standard deviations for these two variables and their correlation.

I found some data here which I converted into a CSV file. If you want to follow along, download the file heights_and_weights.csv. Here's how the scatterplot looks:

```
handw <- read_csv("data/heights_and_weights.csv", col_types = "dd")

ggplot(handw, aes(height_in, weight_lbs)) +
  geom_point(alpha = .2) +
  labs(x = "height (inches)", y = "weight (pounds)")
```

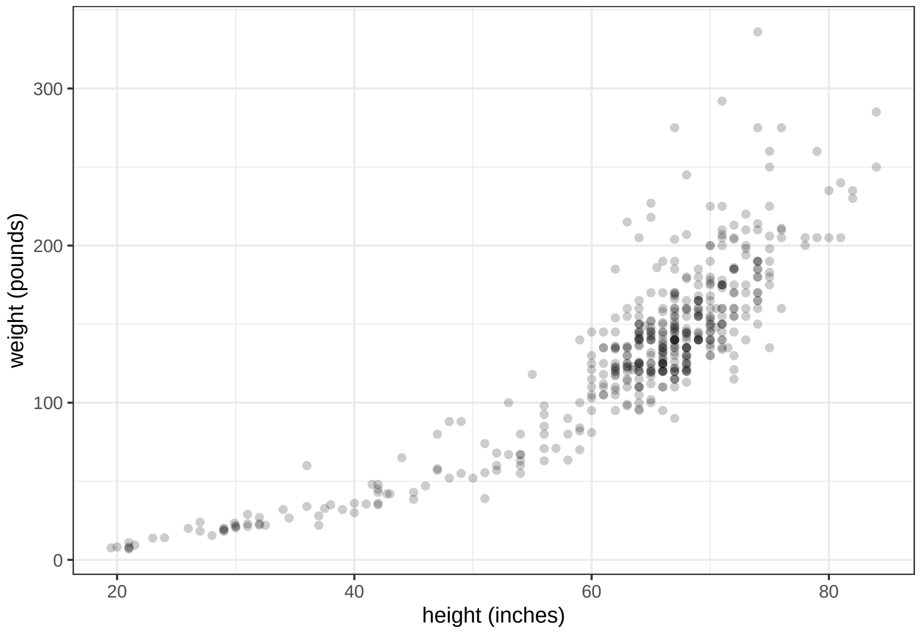

Figure 2.5: Heights and weights of 475 humans (including infants)

Now, that's not quite a linear relationship. We can make it into one by log transforming each of the variables first.

```
handw_log <- handw %>%
  mutate(hlog = log(height_in),
         wlog = log(weight_lbs))
```

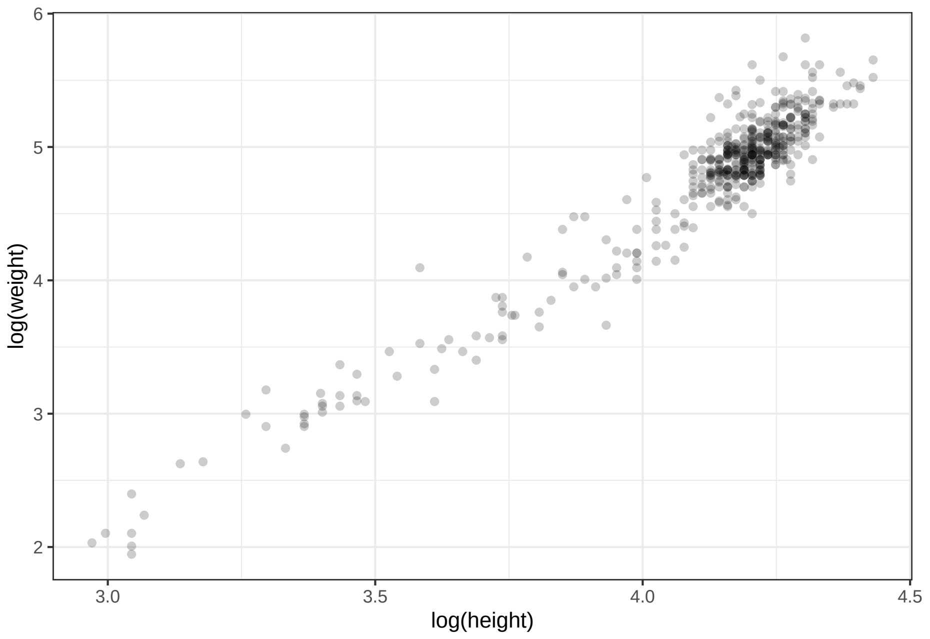

Figure 2.6: Log transformed heights and weights.

The fact that there is a big cluster of points in the top right tail of the cloud probably indicates that there were more adults than kids in this sample, since adults are taller and heavier.

The mean log height is 4.11 (SD = 0.26), while the mean log weight is 4.74 (SD = 0.65). The correlation between log height and log weight, which we can get using the `cor()` function, is very high, 0.96.

We now have all the information we need to simulate the heights and weights of, let's say, 500 humans. But how do we get this information into `MASS::mvrnorm()`? We know the first part of the function call will be `MASS::mvrnorm(500, c(4.11, 4.74), ...)`, but what about `Sigma`, the covariance matrix? We know from above that $\hat{\sigma}_x = 0.26$ and $\hat{\sigma}_y = 0.65$, and $\hat{\rho}_{xy} = 0.96$.

A covariance matrix representating `Sigma` ( ) for bivariate data has the following format:

$$= \begin{pmatrix} {\sigma_x}^2 & \rho_{xy}\sigma_x\sigma_y \\ \rho_{yx}\sigma_y\sigma_x & {\sigma_y}^2 \end{pmatrix}$$

The variances (squared standard deviations, ${\sigma_x}^2$ and ${\sigma_y}^2$) are in the diagonal, and the covariances (the correlation times the two standard deviations, $\rho_{xy}\sigma_x\sigma_y$)

are in the off-diagonal. It is useful to remember that the covariance is just the correlation times the product of the two standard deviations. As we saw above with the correlation matrices, there is redundant information in the table; namely, the covariance appears in the top right cell as well as the bottom left cell of the matrix.

So plugging in the values we got above, our covariance matrix should be

$$\begin{pmatrix} .26^2 & (.96)(.26)(.65) \\ (.96)(.65)(.26) & .65^2 \end{pmatrix} = \begin{pmatrix} .067 & .162 \\ .162 & .423 \end{pmatrix}$$

OK, how do we form `Sigma` in R so that we can pass it to the `mvrnorm()` function? We will use the `matrix()` function, as shown below.

First let's define our covariance and store it in the variable `my_cov`.

```
my_cov <- .96 * .26 * .65
```

Now we'll use `matrix()` to define our `Sigma`, `my_Sigma`.

```
my_Sigma <- matrix(c(.26^2, my_cov, my_cov, .65^2), ncol = 2)
my_Sigma
```

```
##         [,1]    [,2]
## [1,] 0.06760 0.16224
## [2,] 0.16224 0.42250
```

Confused about the `matrix()` function?

The first argument is a vector of values, which we created above using `c()`. The `ncol` argument specifies how many columns the matrix should have. There is also an `nrow` argument that you could use, but if you give it one of the dimensions, it can infer the size of the other using the length of the vector supplied as the first argument.

You can see that `matrix()` fills in the elements of the matrix column by column, rather than row by row by running the following code:

```
matrix(c("a", "b", "c", "d"), ncol = 2)
```

If you want to change this behavior, set the `byrow` argument to `TRUE`.

```
matrix(c("a", "b", "c", "d"), ncol = 2, byrow = TRUE)
```

Great. Now that we've got `my_Sigma`, we're ready to use `MASS::mvrnorm()`. Let's test it out by creating 6 synthetic humans.

```
set.seed(62) # for reproducibility

# passing the *named* vector c(height = 4.11, weight = 4.74)
# for mu gives us column names in the output
log_ht_wt <- MASS::mvrnorm(6,
                           c(height = 4.11, weight = 4.74),
                           my_Sigma)

log_ht_wt
```

```
##        height   weight
## [1,] 4.254209 5.282913
## [2,] 4.257828 4.895222
## [3,] 3.722376 3.759767
## [4,] 4.191287 4.764229
## [5,] 4.739967 6.185191
## [6,] 4.058105 4.806485
```

So `MASS::mvrnorm()` returns a matrix with a row for each simulated human, with the first column representing the log height and the second column representing the log weight. But log heights and log weights are not very useful to us, so let's transform them back by using `exp()`, which is the inverse of the `log()` transform.

```
exp(log_ht_wt)
```

```
##         height    weight
## [1,]  70.40108 196.94276
## [2,]  70.65632 133.64963
## [3,]  41.36254  42.93844
## [4,]  66.10779 117.24065
## [5,] 114.43045 485.50576
## [6,]  57.86453 122.30092
```

So our first simulated human is 70.4 inches tall (about 5'5" or X) and weighs 196.94 pounds (89.32 kg). Sounds about right! (Note also that it will generate observations outside of our original data; we'll get super tall humans, like observation 5, but at least the weight/height relationship will be preserved.)

OK, let's randomly generate a bunch of humans, transform them from log to inches and pounds, and plot them against our original data to see how we're doing.

```
## simulate new humans
new_humans <- MASS::mvrnorm(500,
                            c(height_in = 4.11, weight_lbs = 4.74),
                            my_Sigma) %>%
  exp() %>% # back-transform from log to inches and pounds
  as_tibble() %>% # make tibble for plotting
  mutate(type = "simulated") # tag them as simulated

## combine real and simulated datasets
## handw is variable containing data from heights_and_weights.csv
```

```r
alldata <- bind_rows(handw %>% mutate(type = "real"),
                     new_humans)

ggplot(alldata, aes(height_in, weight_lbs)) +
  geom_point(aes(colour = type), alpha = .1)
```

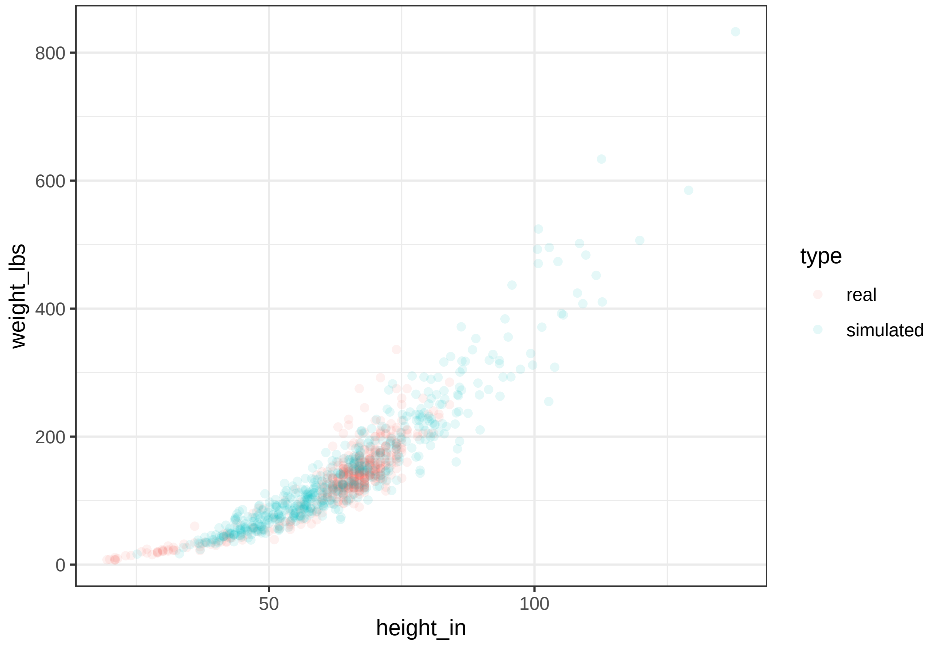

Figure 2.7: Real and simulated humans.

You can see that our simulated humans are much like the normal ones, except that we are creating humans outside the normal range of heights and weights.

2.2.1 Bivariate app

Here is a web app that you should play around with to better understand the relationship between covariance matrices, bivariate normal distributions, and correlation coefficients.

2.3 Relationship between correlation and regression

OK, we know how to estimate correlations, but what if we wanted to predict the weight of someone given their height? This might sound like an impractical problem, but in fact, emergency medical technicians can use this technique to get a quick estimate of people's weights in emergency situations where they need to administer drugs or procedures whose safety depends on the patient's weight, and don't have time to weigh them.

Recall that the GLM for a simple regression model is

$$Y_i = \beta_0 + \beta_1 X_i + e_i.$$

Here, we are trying to predict the weight (Y_i) of person i from their observed height (X_i). In this equation, β_0 and β_1 are the y-intercept and slope parameters respectively, and the e_is are the residuals. It is conventionally assumed that the e_i values are from a normal distribution with mean of zero and variance σ^2; the math-y way of saying this is $e_i \sim N(0, \sigma^2)$, where $\sim$ is read as "distributed according to" and $N(0, \sigma^2)$ means "Normal distribution (N) with mean of 0 and variance of σ^2".

It turns out that if we have estimates of the means of X and Y (denoted by μ_x and μ_y respectively), of their standard deviations ($\hat{\sigma}_x$ and $\hat{\sigma}_y$), and the correlation between X and Y ($\hat{\rho}$), we have all the information we need to estimate the parameters of the regression equation β_0 and β_1. Here's how.

First, the slope of the regression line β_1 equals the correlation coefficient ρ times the ratio of the standard deviations of Y and X.

$$\beta_1 = \rho \frac{\sigma_Y}{\sigma_X}$$

Given the estimates above for log height and weight, can you solve for β_1?

```r
b1 <- .96 * (.65 / .26)
b1
```

```
## [1] 2.4
```

The next thing to note is that for mathematical reasons, the regression line is guaranteed to go through the point corresponding to the mean of X and the mean of Y, i.e., the point (μ_x, μ_y). (You can think of the regression line "pivoting" around that point depending on the slope). You also know that β_0 is the y-intercept, the point where the line crosses the vertical axis at $X = 0$. From this information, and the estimates above, can you figure out the value of β_0?

Here is the reasoning by which you can solve for β_0.

The β_1 value tells you that for each change in X you have a corresponding change of 2.4 for Y, and you know that the line goes through the points (μ_x, μ_y) as well as the y-intercept $(0, \beta_0)$.

Think about stepping back unit-by-unit from $X = \mu_x$ to $X = 0$. At $X = \mu_x$, $Y = 4.74$. Each unit step you take backward in the X dimension, Y will drop by $\beta_1 = 2.4$ units. When you get to zero, Y will have dropped from μ_y to $\mu_y - \mu_x\beta_1$.

So the general solution is: $\beta_0 = \mu_y - \mu_x\beta_1$.

Since $\beta_1 = 2.4$, $\mu_x = 4.11$, and $\mu_y = 4.74$, $\beta_0 = -5.124$. Thus, our regression equation is:

$$Y_i = -5.124 + 2.4X_i + e_i.$$

To check our results, let's first run a regression on the log-transformed data using `lm()`, which estimates parameters using ordinary least squares regression.

```r
summary(lm(wlog ~ hlog, handw_log))
```

```
##
## Call:
## lm(formula = wlog ~ hlog, data = handw_log)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.63296 -0.09915 -0.01366  0.09285  0.65635
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -5.26977    0.13169  -40.02   <2e-16 ***
## hlog         2.43304    0.03194   76.17   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1774 on 473 degrees of freedom
## Multiple R-squared:  0.9246,Adjusted R-squared:  0.9245
## F-statistic:  5802 on 1 and 473 DF,  p-value: < 2.2e-16
```

Looks pretty close. The reason that it doesn't match exactly is only because we've rounded off our estimates to two decimal places for convenience.

As another check, let's superimpose the regression line we computed by hand on the scatterplot of the log-transformed data.

```r
ggplot(handw_log, aes(hlog, wlog)) +
  geom_point(alpha = .2) +
  labs(x = "log(height)", y = "log(weight)") +
  geom_abline(intercept = -5.124, slope = 2.4, colour = 'blue')
```

Looks right.

To close, here are a few implications from the relationship between correlation and regression.

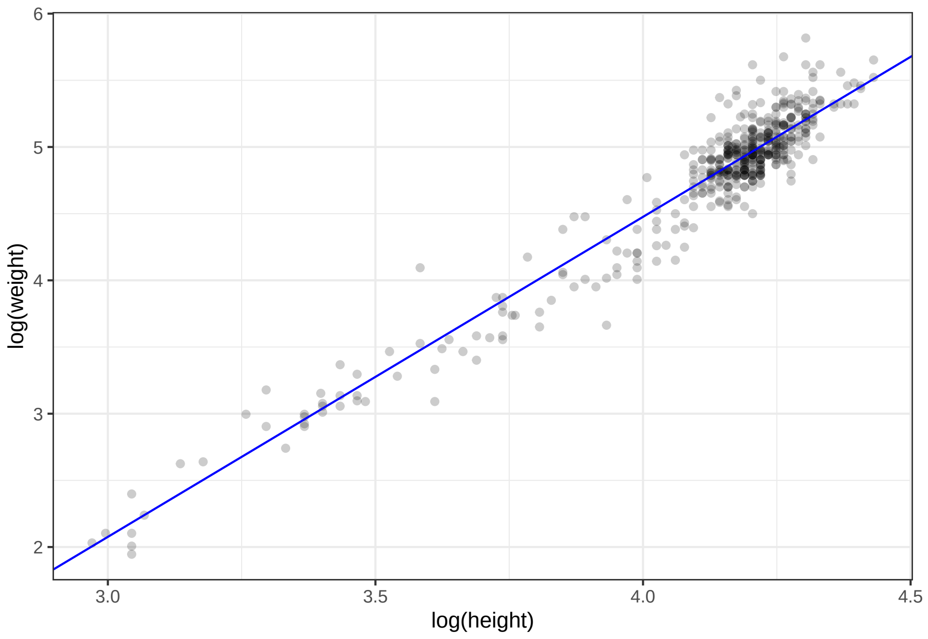

Figure 2.8: Log transformed values with superimposed regression line.

- $\beta_1 = 0$ is the same as $\rho = 0$.
- $\beta_1 > 0$ implies $\rho > 0$, since standard deviations can't be negative.
- $\beta_1 < 0$ implies $\rho < 0$, for the same reason.
- Rejecting the null hypothesis that $\beta_1 = 0$ is the same as rejecting the null hypothesis that $\rho = 0$. The p-values that you get for β_1 in `lm()` will be the same as the one you get for ρ from `cor.test()`.

Chapter 3

Multiple regression

General model for single-level data with m predictors:

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + ... + \beta_m X_{mi} + e_i$$

The individual X_{hi} variables can be any combination of continuous and/or categorical predictors, including interactions among variables.

The β values are referred to as **regression coefficients**. Each β_h is interpreted as the **partial effect of β_h holding constant all other predictor variables.** If you have m predictor variables, you have $m + 1$ regression coefficients: one for the intercept, and one for each predictor.

Although discussions of multiple regression are common in statistical textbooks, you will rarely be able to apply the exact model above. This is because the above model assumes single-level data, whereas most psychological data is multi-level. However, the fundamentals are the same for both types of datasets, so it is worthwhile learning them for the simpler case first.

3.1 An example: How to get a good grade in statistics

Let's look at some (made up, but realistic) data to see how we can use multiple regression to answer various study questions. In this hypothetical study, you have a dataset for 100 statistics students, which includes their final course grade (`grade`), the number of lectures each student attended (`lecture`, an integer ranging from 0-10), how many times each student clicked to download online materials (`nclicks`) and each student's grade point average prior to taking the course, `GPA`, which ranges from 0 (fail) to 4 (highest possible grade).

3.1.1 Data import and visualization

Let's load in the data grades.csv and have a look.

```
library("corrr") # correlation matrices
library("tidyverse")

grades <- read_csv("data/grades.csv", col_types = "ddii")

grades
```

```
## # A tibble: 100 x 4
##    grade   GPA lecture nclicks
##    <dbl> <dbl>   <int>   <int>
##  1  2.40 1.13        6      88
##  2  3.67 0.971       6      96
##  3  2.85 3.34        6     123
##  4  1.36 2.76        9      99
##  5  2.31 1.02        4      66
##  6  2.58 0.841       8      99
##  7  2.69 4           5      86
##  8  3.05 2.29        7     118
##  9  3.21 3.39        9      98
## 10  2.24 3.27       10     115
## # ... with 90 more rows
```

First let's look at all the pairwise correlations.

```
grades %>%
  correlate() %>%
  shave() %>%
  fashion()
```

```
##
## Correlation method: 'pearson'
## Missing treated using: 'pairwise.complete.obs'
```

```
##   rowname grade  GPA lecture nclicks
## 1   grade
## 2     GPA   .25
## 3 lecture   .24  .44
## 4 nclicks   .16  .30     .36
```

```
pairs(grades)
```

3.1.2 Estimation and interpretation

To estimate the regression coefficients (the βs), we will use the `lm()` function. For a GLM with m predictors:

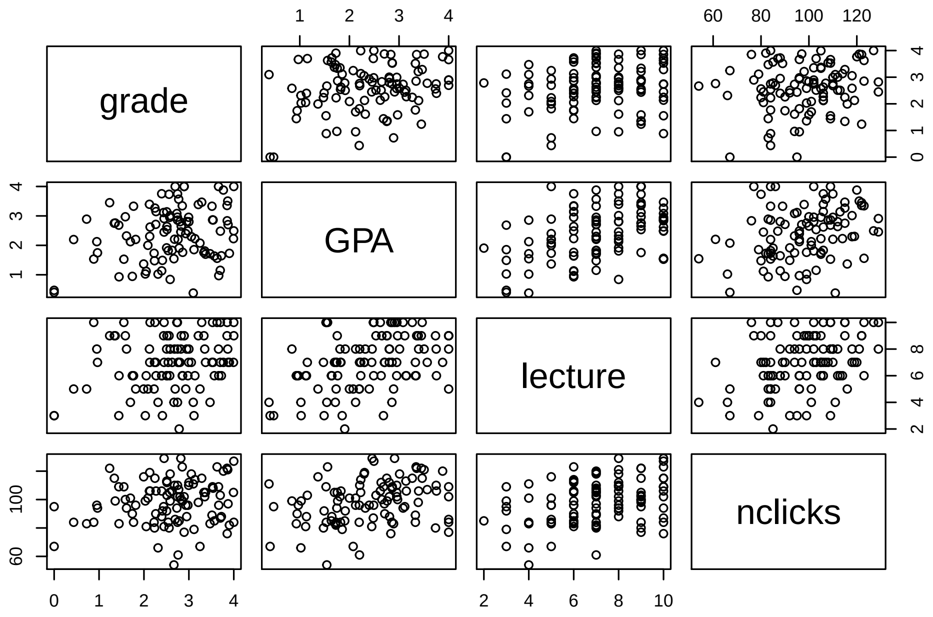

Figure 3.1: All pairwise relationships in the 'grades' dataset.

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + ... + \beta_m X_{mi} + e_i$$

The call to base R's `lm()` is

`lm(Y ~ X1 + X2 + ... + Xm, data)`

The `Y` variable is your response variable, and the `X` variables are the predictor variables. Note that you don't need to explicitly specify the intercept or residual terms; those are included by default.

For the current data, let's predict `grade` from `lecture` and `nclicks`.

```
my_model <- lm(grade ~ lecture + nclicks, grades)

summary(my_model)
```

```
##
## Call:
## lm(formula = grade ~ lecture + nclicks, data = grades)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.21653 -0.40603  0.02267  0.60720  1.38558
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1.462037   0.571124   2.560   0.0120 *
## lecture     0.091501   0.045766   1.999   0.0484 *
## nclicks     0.005052   0.006051   0.835   0.4058
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8692 on 97 degrees of freedom
## Multiple R-squared:  0.06543,Adjusted R-squared:  0.04616
## F-statistic: 3.395 on 2 and 97 DF,  p-value: 0.03756
```

We'll often write the parameter symbol with a little hat on top to make clear that we are dealing with estimates from the sample rather than the (unknown) true population values. From above:

- $\hat{\beta}_0 = 1.46$
- $\hat{\beta}_1 = 0.09$
- $\hat{\beta}_2 = 0.01$

This tells us that a person's predicted grade is related to their lecture attendance and download rate by the following formula:

`grade` $= 1.46 + 0.09 \times$ `lecture` $+ 0.01 \times$ `nclicks`

Because $\hat{\beta}_1$ and $\hat{\beta}_2$ are both positive, we know that higher values of `lecture` and `nclicks` are associated with higher grades.

So if you were asked, what grade would you predict for a student who attends 3 lectures and downloaded 70 times, you could easily figure that out by substituting the appropriate values.

`grade` $= 1.46 + 0.09 \times 3 + 0.01 \times 70$

which equals

`grade` $= 1.46 + 0.27 + 0.7$

and reduces to

`grade` $= 2.43$

3.1.3 Predictions from the linear model using `predict()`

If we want to predict response values for new predictor values, we can use the `predict()` function in base R.

`predict()` takes two main arguments. The first argument is a fitted model object (i.e., `my_model` from above) and the second is a data frame (or tibble) containing new values for the predictors.

You need to include **all** of the predictor variables in the new table. You'll get an error message if your tibble is missing any predictors. You also need to make sure that the variable names in the new table **exactly** match the variable names in the model.

Let's create a tibble with new values and try it out.

```
## a 'tribble' is a way to make a tibble by rows,
## rather than by columns. This is sometimes useful
new_data <- tribble(~lecture, ~nclicks,
                    3, 70,
                    10, 130,
                    0, 20,
                    5, 100)
```

The `tribble()` function provides a way to build a tibble row by row, whereas with `tibble()` the table is built column by column.

The first row of the `tribble()` contains the column names, each preceded by a tilde (`~`).

This is sometimes easier to read than doing it row by row, although the result is the same. Consider that we could have made the above table using

```
new_data <- tibble(lecture = c(3, 10, 0, 5),
                   nclicks = c(70, 130, 20, 100))
```

Now that we've created our table `new_data`, we just pass it to `predict()` and it will return a vector with the predictions for Y (`grade`).

```
predict(my_model, new_data)
```

```
##        1        2        3        4 
## 2.090214 3.033869 1.563087 2.424790
```

That's great, but maybe we want to line it up with the predictor values. We can do this by just adding it as a new column to `new_data`.

```
new_data %>%
  mutate(predicted_grade = predict(my_model, new_data))
```

```
## # A tibble: 4 x 3
##   lecture nclicks predicted_grade
##     <dbl>   <dbl>           <dbl>
## 1       3      70            2.09
## 2      10     130            3.03
## 3       0      20            1.56
## 4       5     100            2.42
```

Want to see more options for `predict()`? Check the help at `?predict.lm`.

3.1.4 Visualizing partial effects

As noted above the parameter estimates for each regression coefficient tell us about the **partial** effect of that variable; it's effect holding all of the others constant. Is there a way to visualize this partial effect? Yes, you can do this using the `predict()` function, by making a table with varying values for the focal predictor, while filling in all of the other predictors with their mean values.

For example, let's visualize the partial effect of `lecture` on `grade` holding `nclicks` constant at its mean value.

```
nclicks_mean <- grades %>% pull(nclicks) %>% mean()

## new data for prediction
new_lecture <- tibble(lecture = 0:10,
                      nclicks = nclicks_mean)

## add the predicted value to new_lecture
new_lecture2 <- new_lecture %>%
  mutate(grade = predict(my_model, new_lecture))

new_lecture2
```

```
## # A tibble: 11 x 3
##    lecture nclicks grade
##      <int>   <dbl> <dbl>
##  1       0    98.3  1.96
##  2       1    98.3  2.05
##  3       2    98.3  2.14
##  4       3    98.3  2.23
##  5       4    98.3  2.32
##  6       5    98.3  2.42
##  7       6    98.3  2.51
##  8       7    98.3  2.60
##  9       8    98.3  2.69
## 10       9    98.3  2.78
## 11      10    98.3  2.87
```

Now let's plot.

```
ggplot(grades, aes(lecture, grade)) +
  geom_point() +
  geom_line(data = new_lecture2)
```

Partial effect plots only make sense when there are no interactions in the model between the focal predictor and any other predictor.

The reason is that when there are interactions, the partial effect of focal predictor X_i will differ across the values of the other variables it interacts with.

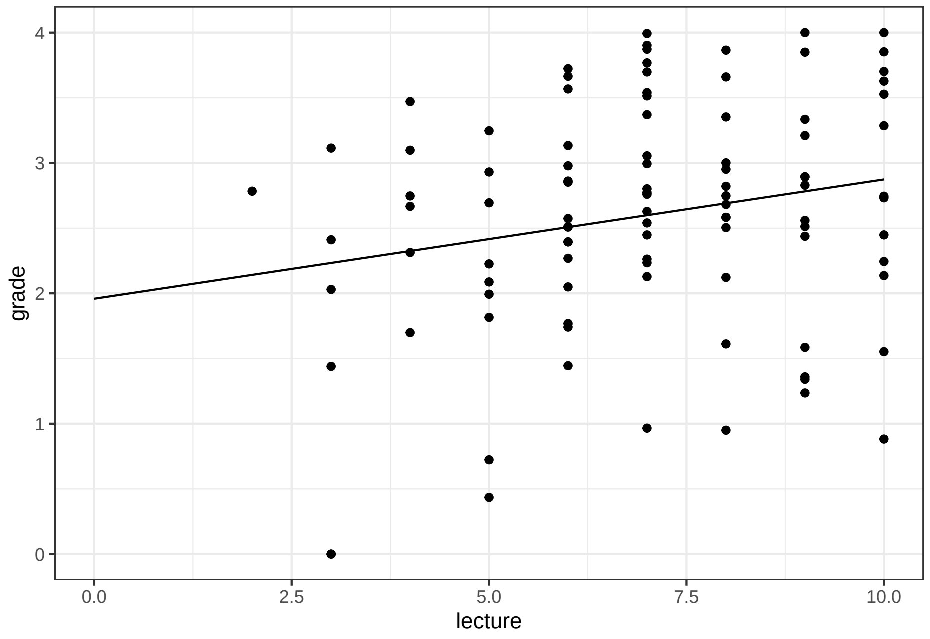

Figure 3.2: Partial effect of ’lecture’ on grade, with nclicks at its mean value.

Now can you visualize the partial effect of `nclicks` on `grade`?

See the solution at the bottom of the page.

3.1.5 Standardizing coefficients

One kind of question that we often use multiple regression to address is, **Which predictors matter most in predicting Y?**

Now, you can’t just read off the $\hat{\beta}$ values and choose the one with the largest absolute value, because the predictors are all on different scales. To answer this question, you need to **center** and **scale** the predictors.

Remember z scores?

$$z = \frac{X - \mu_x}{\sigma_x}$$

A z score represents the distance of a score X from the sample mean (μ_x) in standard deviation units (σ_x). So a z score of 1 means that the score is one standard deviation about the mean; a z-score of -2.5 means 2.5 standard deviations below the mean. Z-scores give us a way of comparing things that come from different populations by calibrating them to the standard normal distribution (a distribution with a mean of 0 and a standard deviation of 1).

So we re-scale our predictors by converting them to z-scores. This is easy enough to do.

```
grades2 <- grades %>%
  mutate(lecture_c = (lecture - mean(lecture)) / sd(lecture),
         nclicks_c = (nclicks - mean(nclicks)) / sd(nclicks))

grades2
```

```
## # A tibble: 100 x 6
##    grade   GPA lecture nclicks lecture_c nclicks_c
##    <dbl> <dbl>   <int>   <int>     <dbl>     <dbl>
##  1  2.40 1.13        6      88  -0.484     -0.666
##  2  3.67 0.971       6      96  -0.484     -0.150
##  3  2.85 3.34        6     123  -0.484      1.59
##  4  1.36 2.76        9      99   0.982      0.0439
##  5  2.31 1.02        4      66  -1.46      -2.09
##  6  2.58 0.841       8      99   0.493      0.0439
##  7  2.69 4           5      86  -0.972     -0.796
##  8  3.05 2.29        7     118   0.00488    1.27
##  9  3.21 3.39        9      98   0.982     -0.0207
## 10  2.24 3.27       10     115   1.47       1.08
## # ... with 90 more rows
```

Now let's re-fit the model using the centered and scaled predictors.

```
my_model_scaled <- lm(grade ~ lecture_c + nclicks_c, grades2)

summary(my_model_scaled)
```

```
##
## Call:
## lm(formula = grade ~ lecture_c + nclicks_c, data = grades2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.21653 -0.40603  0.02267  0.60720  1.38558
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.59839    0.08692  29.895   <2e-16 ***
## lecture_c    0.18734    0.09370   1.999   0.0484 *
## nclicks_c    0.07823    0.09370   0.835   0.4058
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.8692 on 97 degrees of freedom
## Multiple R-squared:  0.06543,Adjusted R-squared:  0.04616
## F-statistic: 3.395 on 2 and 97 DF,  p-value: 0.03756
```

This tells us that lecture_c has a relatively larger influence; for each standard deviation increase in this variable, `grade` increases by about 0.19.

3.1.6 Model comparison

Another common kind of question multiple regression is also used to address is of the form: Does some predictor or set of predictors of interest significantly impact my response variable **over and above the effects of some control variables**?

For example, we saw above that the model including `lecture` and `nclicks` was statistically significant, $F(2, 97) = 3.395$, $p = 0.038$.

The null hypothesis for a regression model with m predictors is

$$H_0 : \beta_1 = \beta_2 = ... = \beta_m = 0;$$

in other words, that all of the coefficients (except the intercept) are zero. If the null hypothesis is true, then the null model

$$Y_i = \beta_0$$

gives just as good of a prediction as the model including all of the predictors and their coefficients. In other words, your best prediction for Y is just its mean (μ_y); the X variables are irrelevant. We rejected this null hypothesis, which implies that we can do better by including our two predictors, `lecture` and `nclicks`.

But you might ask: maybe its the case that better students get better grades, and the relationship between `lecture`, `nclicks`, and `grade` is just mediated by student quality. After all, better students are more likely to go to lecture and download the materials. So we can ask, are attendance and downloads associated with better grades **above and beyond** student ability, as measured by GPA?

The way we can test this hypothesis is by using **model comparison**. The logic is as follows. First, estimate a model containing any control predictors but excluding the focal predictors of interest. Second, estimate a model containing the control predictors as well as the focal predictors. Finally, compare the two models, to see if there is any statistically significant gain by including the predictors.

Here is how you do this:

```r
m1 <- lm(grade ~ GPA, grades) # control model
m2 <- lm(grade ~ GPA + lecture + nclicks, grades) # bigger model

anova(m1, m2)
```

```
## Analysis of Variance Table
##
## Model 1: grade ~ GPA
## Model 2: grade ~ GPA + lecture + nclicks
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     98 73.528
## 2     96 71.578  2    1.9499 1.3076 0.2752
```

The null hypothesis is that we are just as good predicting `grade` from `GPA` as we are predicting it from `GPA` plus `lecture` and `nclicks`. We will reject the null if adding these two variables leads to a substantial enough reduction in the **residual sums of squares** (RSS); i.e., if they explain away enough residual variance.

We see that this is not the case: $F(2, 96) = 1.308$, $p = 0.275$. So we don't have evidence that lecture attendance and downloading the online materials is associated with better grades above and beyond student ability, as measured by GPA.

3.2 Dealing with categorical predictors

You can include categorical predictors in a regression model, but first you have to code them as numerical variables. There are a couple of important considerations here.

A **nominal** variable is a categorical variable for which there is no inherent ordering among the levels of the variable. Pet ownership (cat, dog, ferret) is a nominal variable; cat is not greater than dog and dog is not greater than ferret.

It is common to code nominal variables using numbers. However, you have to be **very careful** about using numerically-coded nominal variables in your models. If you have a number that is really just a nominal variable, make sure you define it as type `factor()` before entering it into the model. Otherwise, it will try to treat it as an actual number, and the results of your modeling will be garbage!

It is far too easy to make this mistake, and difficult to catch if authors do not share their data and code. In 2016, a paper on religious affiliation and altruism in children that was published in Current Biology had to be retracted for just this kind of mistake.

3.2.1 Dummy coding

For a factor with two levels, choose one level as zero and the other as one. The choice is arbitrary, and will affect the sign of the coefficient, but not its standard error or p-value. Here is some code that will do this. Note that if you have a predictor of type character or factor, R will automatically do that for you. We don't want R to do this for reasons that will become apparent in the next lecture, so let's learn how to make our own numeric predictor.

First, we gin up some fake data to use in our analysis.

```
fake_data <- tibble(Y = rnorm(10),
                    group = rep(c("A", "B"), each = 5))

fake_data
```

```
## # A tibble: 10 x 2
##          Y group
##      <dbl> <chr>
##  1  0.847  A
##  2 -1.36   A
##  3  0.280  A
##  4 -0.459  A
##  5  0.0105 A
##  6  0.975  B
##  7 -0.568  B
##  8  0.707  B
##  9  0.728  B
## 10  0.311  B
```

Now let's add a new variable, `group_d`, which is the dummy coded group variable. We will use the `dplyr::if_else()` function to define the new column.

```
fake_data2 <- fake_data %>%
  mutate(group_d = if_else(group == "B", 1, 0))

fake_data2
```

```
## # A tibble: 10 x 3
##          Y group group_d
##      <dbl> <chr>   <dbl>
##  1  0.847  A           0
##  2 -1.36   A           0
##  3  0.280  A           0
##  4 -0.459  A           0
##  5  0.0105 A           0
##  6  0.975  B           1
##  7 -0.568  B           1
##  8  0.707  B           1
##  9  0.728  B           1
## 10  0.311  B           1
```

Now we just run it as a regular regression model.

```
summary(lm(Y ~ group_d, fake_data2))
```

```
##
## Call:
## lm(formula = Y ~ group_d, data = fake_data2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2222 -0.2724  0.2113  0.3863  0.9834
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.1360     0.3253  -0.418    0.687
## group_d       0.5665     0.4600   1.232    0.253
##
## Residual standard error: 0.7273 on 8 degrees of freedom
## Multiple R-squared:  0.1594,Adjusted R-squared:  0.05429
## F-statistic: 1.517 on 1 and 8 DF,  p-value: 0.2531
```

Note that if we reverse the coding we get the same result, just the sign is different.

```
fake_data3 <- fake_data %>%
  mutate(group_d = if_else(group == "A", 1, 0))

summary(lm(Y ~ group_d, fake_data3))
```

```
##
## Call:
## lm(formula = Y ~ group_d, data = fake_data3)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2222 -0.2724  0.2113  0.3863  0.9834
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   0.4306     0.3253   1.324    0.222
## group_d      -0.5665     0.4600  -1.232    0.253
##
## Residual standard error: 0.7273 on 8 degrees of freedom
## Multiple R-squared:  0.1594,Adjusted R-squared:  0.05429
## F-statistic: 1.517 on 1 and 8 DF,  p-value: 0.2531
```

The interpretation of the intercept is the estimated mean for the group coded as zero. You can see by plugging in zero for X in the prediction formula below. Thus, β_1 can be interpreted as the difference between the mean for the baseline group and the group coded as 1.

$$\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i$$

3.2.2 Dummy coding when $k > 2$

When the predictor variable is a factor with k levels where $k > 2$, we need $k-1$ predictors to code that variable. So if the factor has 4 levels, we'll need to define three predictors. Here is code to do that. Try it out and see if you can figure out how it works.

```
mydata <- tibble(season = rep(c("winter", "spring", "summer", "fall"), each = 5),
                 bodyweight_kg = c(rnorm(5, 105, 3),
                                   rnorm(5, 103, 3),
                                   rnorm(5, 101, 3),
                                   rnorm(5, 102.5, 3)))

mydata
```

```
## # A tibble: 20 x 2
##    season bodyweight_kg
##    <chr>          <dbl>
##  1 winter         105.
##  2 winter         106.
##  3 winter         104.
##  4 winter         107.
##  5 winter         104.
##  6 spring         106.
##  7 spring          98.8
##  8 spring          97.6
##  9 spring         105.
## 10 spring          98.0
## 11 summer         104.
## 12 summer         106.
## 13 summer         103.
## 14 summer          95.0
## 15 summer         101.
## 16 fall           104.
## 17 fall           107.
## 18 fall           101.
## 19 fall            99.5
## 20 fall           105.
```

Now let's add three predictors to code the variable `season`.

```
## baseline value is 'winter'
mydata2 <- mydata %>%
  mutate(V1 = if_else(season == "spring", 1, 0),
         V2 = if_else(season == "summer", 1, 0),
         V3 = if_else(season == "fall", 1, 0))

mydata2
```

```
## # A tibble: 20 x 5
##    season bodyweight_kg    V1    V2    V3
##    <chr>          <dbl> <dbl> <dbl> <dbl>
##  1 winter         105.      0     0     0
##  2 winter         106.      0     0     0
##  3 winter         104.      0     0     0
##  4 winter         107.      0     0     0
##  5 winter         104.      0     0     0
##  6 spring         106.      1     0     0
##  7 spring          98.8     1     0     0
##  8 spring          97.6     1     0     0
##  9 spring         105.      1     0     0
## 10 spring          98.0     1     0     0
## 11 summer         104.      0     1     0
## 12 summer         106.      0     1     0
## 13 summer         103.      0     1     0
## 14 summer          95.0     0     1     0
## 15 summer         101.      0     1     0
## 16 fall           104.      0     0     1
## 17 fall           107.      0     0     1
## 18 fall           101.      0     0     1
## 19 fall            99.5     0     0     1
## 20 fall           105.      0     0     1
```

3.3 Equivalence between multiple regression and one-way ANOVA

If we wanted to see whether our bodyweight varies over season, we could do a one way ANOVA on `mydata2` like so.

```
## make season into a factor with baseline level 'winter'
mydata3 <- mydata2 %>%
  mutate(season = factor(season, levels = c("winter", "spring", "summer", "fall")))

my_anova <- aov(bodyweight_kg ~ season, mydata3)
summary(my_anova)
```

```
##             Df Sum Sq Mean Sq F value Pr(>F)
## season       3   47.9   15.97   1.413  0.276
## Residuals   16  180.9   11.30
```

OK, now can we replicate that result using the regression model below?

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \beta_3 X_{3i} + e_i$$

```
summary(lm(bodyweight_kg ~ V1 + V2 + V3, mydata3))
```

```
##
## Call:
## lm(formula = bodyweight_kg ~ V1 + V2 + V3, data = mydata3)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.9150 -2.2612  0.0746  2.2577  4.6067
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  105.129      1.504  69.917   <2e-16 ***
## V1            -4.084      2.126  -1.920   0.0728 .
## V2            -3.250      2.126  -1.529   0.1459
## V3            -1.846      2.126  -0.868   0.3982
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.362 on 16 degrees of freedom
## Multiple R-squared:  0.2094,Adjusted R-squared:  0.06116
## F-statistic: 1.413 on 3 and 16 DF,  p-value: 0.2757
```

Note that the F values and p values are identical for the two methods!

3.4 Solution to partial effect plot

First create a tibble with new predictors. We might also want to know the range of values that `nclicks` varies over.

```
lecture_mean <- grades %>% pull(lecture) %>% mean()
min_nclicks <- grades %>% pull(nclicks) %>% min()
max_nclicks <- grades %>% pull(nclicks) %>% max()

## new data for prediction
new_nclicks <- tibble(lecture = lecture_mean,
                      nclicks = min_nclicks:max_nclicks)
```

```
## add the predicted value to new_lecture
new_nclicks2 <- new_nclicks %>%
  mutate(grade = predict(my_model, new_nclicks))

new_nclicks2
```

```
## # A tibble: 76 x 3
##    lecture nclicks grade
##      <dbl>   <int> <dbl>
##  1    6.99      54  2.37
##  2    6.99      55  2.38
##  3    6.99      56  2.38
##  4    6.99      57  2.39
##  5    6.99      58  2.39
##  6    6.99      59  2.40
##  7    6.99      60  2.40
##  8    6.99      61  2.41
##  9    6.99      62  2.41
## 10    6.99      63  2.42
## # ... with 66 more rows
```

Now plot.

```
ggplot(grades, aes(nclicks, grade)) +
  geom_point() +
  geom_line(data = new_nclicks2)
```

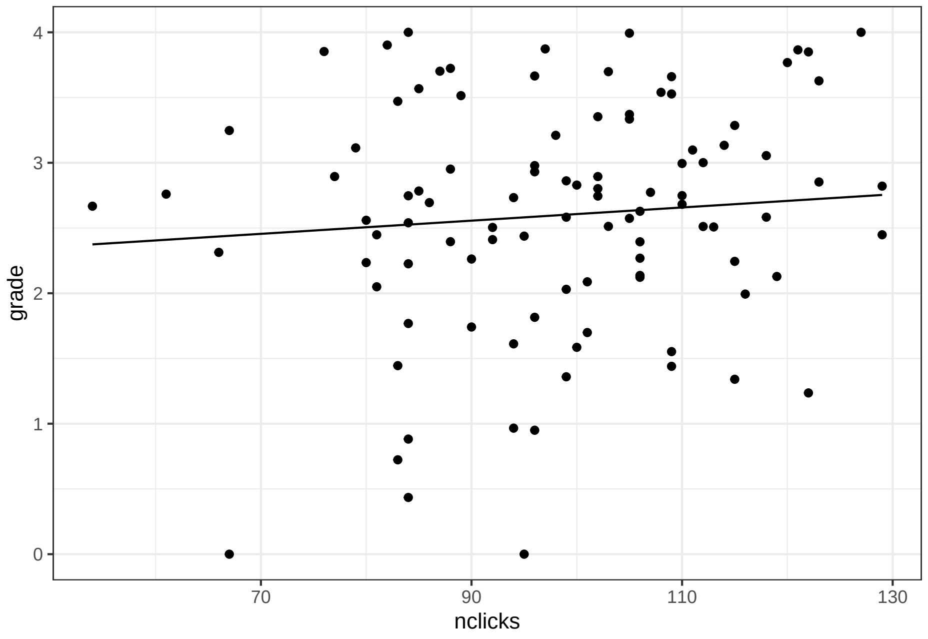

Figure 3.3: Partial effect plot of nclicks on grade.

Chapter 4

Interactions

4.1 Learning objectives

- model and interpret continuous-by-categorical interactions
- model and interpret categorical-by-categorical interactions in factorial designs
- estimate and test effects in factorial designs using ANOVA or regression

4.2 Interactions

Up to now, we've been focusing on estimating and interpreting the effect of a variable or linear combination of predictor variables on a response variable. However, there are often situations where the effect of one predictor on the response depends on the value of another predictor variable. We can actually estimate and interpret this dependency as well, by including an **interaction** term in our model.

4.3 Continuous-by-Categorical Interactions

One common example of this is when you are interested in whether a linear relationship between a continous predictor and a continuous response is different for two groups.

Let's consider a simple fictional example. Say you are interested in the effects of sonic distraction on cognitive performance. Each participant in your study is randomly assigned to receive a particular amount of sonic distraction while they perform a simple reaction time task (respond as quickly as possible to a flashing light). You have a technique that allows you to automatically generate different levels of background noise (e.g., frequency and amplitude of city sounds, such as sirens, jackhammers, people yelling, glass breaking, etc.). Each participant performs the task for a randomly chosen level of distraction (0 to 100). Your hypothesis is that urban living makes people's task performance more immune to sonic distraction. You want to compare the relationship between distraction and performance for city dwellers to the relationship for people from quieter rural environments.

You have three variables:

- A continuous response variable, `mean_RT`, with higher levels reflecting slower RTs;
- A continuous predictor variable, level of sonic distraction (`dist_level`), with higher levels indicating more distraction;
- A factor with two levels, `group` (urban vs. rural).

Let's start by simulating some data for the urban group. Let's assume that with zero distraction (silence), the average RT is about 450 milliseconds, and that with each unit increase on the distraction scale, RT increases about 2 ms. This gives us the following linear model:

$$Y_i = 450 + 2X_i + e_i$$

where X_i is the amount of sonic distraction.

Let's simulate data for 100 participants as below with $\sigma = 80$, setting the seed before we begin.

```r
library("tidyverse")
set.seed(1031)

n_subj <- 100L  # simulate data for 100 subjects
b0_urban <- 450 # y-intercept
b1_urban <- 2   # slope

# decomposition table
urban <- tibble(
  subj_id = 1:n_subj,
  group = "urban",
  b0 = 450,
  b1 = 2,
  dist_level = sample(0:n_subj, n_subj, replace = TRUE),
  err = rnorm(n_subj, mean = 0, sd = 80),
  simple_rt = b0 + b1 * dist_level + err)

urban
```

```
## # A tibble: 100 x 7
##    subj_id group    b0    b1 dist_level     err simple_rt
##        <int> <chr> <dbl> <dbl>      <int>   <dbl>      <dbl>
##   1        1 urban   450     2         59  -36.1        532.
##   2        2 urban   450     2         45  128.         668.
##   3        3 urban   450     2         55   23.5        584.
##   4        4 urban   450     2          8    1.04       467.
##   5        5 urban   450     2         47   48.7        593.
##   6        6 urban   450     2         96   88.2        730.
##   7        7 urban   450     2         62  110.         684.
##   8        8 urban   450     2          8  -91.6        374.
##   9        9 urban   450     2         15 -109.         371.
##  10       10 urban   450     2         70   20.7        611.
## # ... with 90 more rows
```

Let's plot the data we created, along with the line of best fit.

```
ggplot(urban, aes(dist_level, simple_rt)) +
  geom_point(alpha = .2) +
  geom_smooth(method = "lm", se = FALSE)
```

```
## `geom_smooth()` using formula 'y ~ x'
```

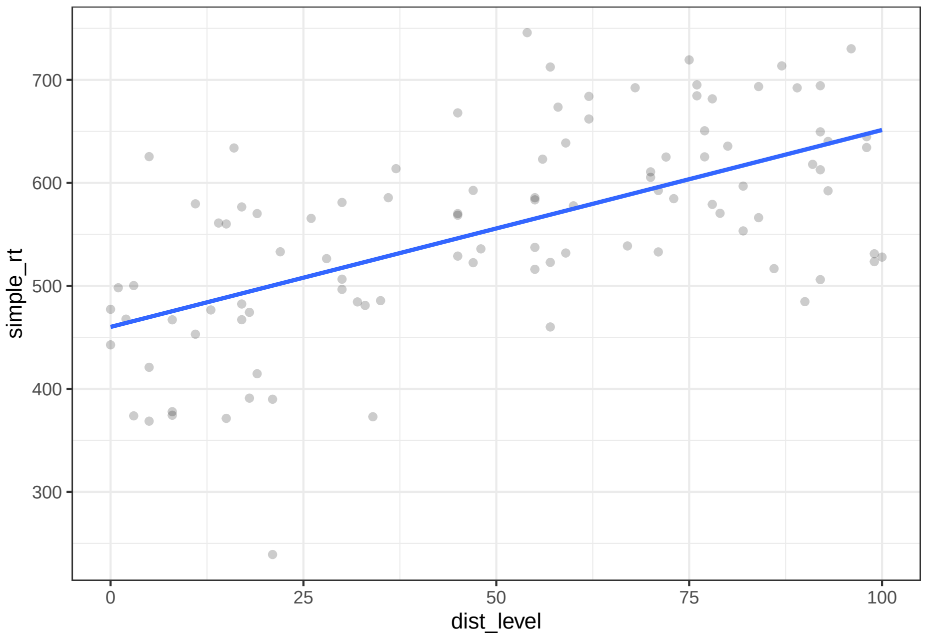

Figure 4.1: *Effect of sonic distraction on simple RT, urban group.*

Now let's simulate data for the urban group. We assume that these participants should perhaps have a little higher intercept, maybe because they are less familiar with technology. Most importantly, we assume that they would have a steeper slope because they are more affected by the noise. Something like:

$$Y_i = 500 + 3X_i + e_i$$

```r
b0_rural <- 500
b1_rural <- 3

rural <- tibble(
  subj_id = 1:n_subj + n_subj,
  group = "rural",
  b0 = b0_rural,
  b1 = b1_rural,
  dist_level = sample(0:n_subj, n_subj, replace = TRUE),
  err = rnorm(n_subj, mean = 0, sd = 80),
  simple_rt = b0 + b1 * dist_level + err)
```

Now let's plot the data from the two groups side by side.

```r
all_data <- bind_rows(urban, rural)

ggplot(all_data %>% mutate(group = fct_relevel(group, "urban")),
       aes(dist_level, simple_rt, colour = group)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ group) +
  theme(legend.position = "none")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

Here we see very clearly the difference in slope that we built into our data. How do we test whether the two slopes are significantly different? To do this, we can't have two separate regressions. We need to bring the two regression lines into the same model. How do we do this?

Note that we can represent one of the regression lines in terms of 'offset' values from the other. We (arbitrarily) choose one group as our 'baseline' group, and represent the y-intercept and slope of the other group as offsets from this baseline. So if we choose the urban group as our baseline, we can express the y-intercept and slope for the rural group in terms of two offsets, β_2 and β_3, for the y-intercept and slope, respectively.

- y-intercept: $\beta_{0\_rural} = \beta_{0\_urban} + \beta_2$
- slope: $\beta_{1\_rural} = \beta_{1\_urban} + \beta_3$

Our urban group had parameters $\beta_{0\_urban} = 450$ (`b0_urban <- 450`) and $\beta_{1\_urban} = 2$ (`b1_urban <- 2`), whereas the rural broup had $\beta_{0\_rural} = 500$ (`b0_rural <- 500`) and $\beta_{1\_rural} = 3$ (`b1_rural <- 3`). It directly follows that $\beta_2 = 50$ and $\beta_3 = 1$ for our simulated example.

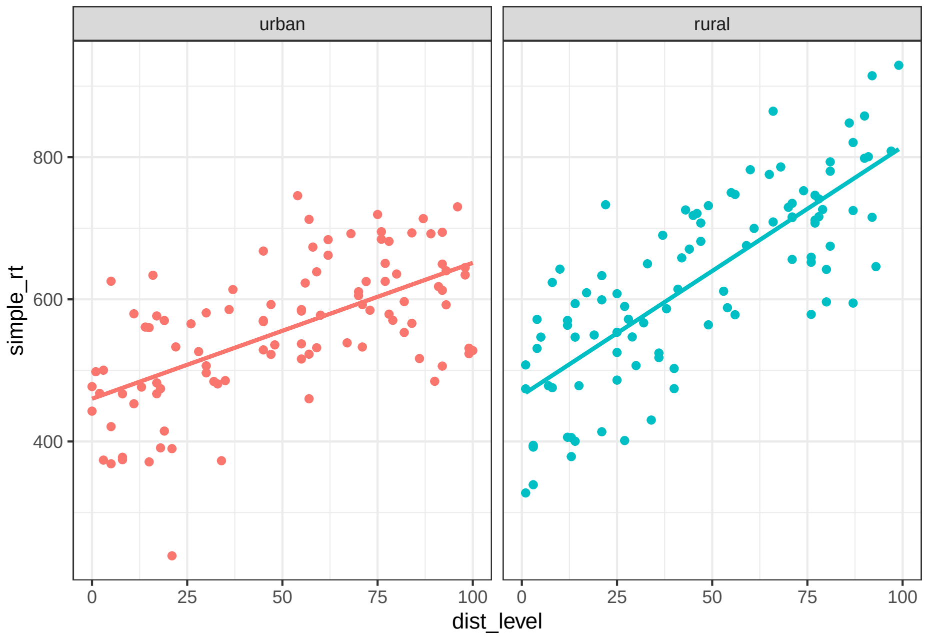

Figure 4.2: *Effect of sonic distraction on simple RT for urban and rural participants.*

So our two regression models are now:

$$Y_{i\_urban} = \beta_{0\_urban} + \beta_{1\_urban} X_i + e_i$$

and

$$Y_{i\_rural} = (\beta_{0\_urban} + \beta_2) + (\beta_{1\_urban} + \beta_3) X_i + e_i.$$

OK, we feel like we're closer to getting these into a single regression model. Here's the final trick. We define an additional dummy predictor variable that takes on the value 0 for the urban group (which we chose as our 'baseline' group) and 1 for the other group. Our final model takes the following form.

Regression model with a continuous-by-categorical interaction.

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 X_{2i} + \beta_3 X_{1i} X_{2i} + e_i$$

where

- X_{1i} is the continous predictor, and

- X_{2i} is a dummy-coded variable taking on 0 for the baseline, 1 for the alternative group.

Interpretation of parameters:

- β_0: y-intercept for the baseline group;
- β_1: slope for the baseline group;
- β_2: offset to y-intercept for the alternative group;
- β_3: offset to slope for the alternative group.

Estimation in R:

`lm(Y ~ X1 + X2 + X1:X2, data)` or, as a shortcut:

`lm(Y ~ X1 * X2)` where `*` means "all possible main effects and interactions of X1 and X2"

The term associated with β_3 is an **interaction term**, where the predictor is the product of predictor values. Let's now show that the above GLM gives us the two regression lines that we want. Plugging in 0 for X_{2i} and reducing gives us

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 0 + \beta_3 X_{1i} 0 + e_i$$

which is just

$$Y_i = \beta_0 + \beta_1 X_{1i} + e_i,$$

the regression equation for our baseline (urban) group. Compare this to Y_{i_urban} above.

Plugging in 1 for X_{2i} should give us the equation for our rural group. We get

$$Y_i = \beta_0 + \beta_1 X_{1i} + \beta_2 1 + \beta_3 X_{1i} 1 + e_i$$

which, after reducing and applying a little algebra, can also be expressed as

$$Y_i = \beta_0 + \beta_2 + (\beta_1 + \beta_3) X_{1i} + e_i.$$

Compare this to Y_{i_rural} above. The dummy-coding trick works!

Now let's see how to estimate the regression in R. Let's say we wanted to test the hypothesis that the slopes for the two lines are different. Note that this just amounts to testing the null hypothesis that $\beta_3 = 0$, because β_3 is our slope offset.

We have already created the dataset `all_data` combining the simulated data for our two groups. The way we express our model using R formula syntax is `Y ~`

`X1 + X2 + X1:X2` where `X1:X2` tells R to create a predictor that is the product of predictors X1 and X2. There is a shortcut `Y ~ X1 * X2` which tells R to calculate all possible main effects and interactions. First we'll add a dummy predictor to our model, storing the result in `all_data2`.

```
all_data2 <- all_data %>%
  mutate(grp = if_else(group == "rural", 1, 0))
```

```
sonic_mod <- lm(simple_rt ~ dist_level + grp + dist_level:grp, all_data2)

summary(sonic_mod)
```

```
##
## Call:
## lm(formula = simple_rt ~ dist_level + grp + dist_level:grp, data = all_data2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -261.130  -50.749    3.617   62.304  191.211
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)    460.1098    15.5053  29.674  < 2e-16 ***
## dist_level       1.9123     0.2620   7.299 7.07e-12 ***
## grp              4.8250    21.7184   0.222    0.824
## dist_level:grp   1.5865     0.3809   4.166 4.65e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 81.14 on 196 degrees of freedom
## Multiple R-squared:  0.5625,Adjusted R-squared:  0.5558
## F-statistic: 83.99 on 3 and 196 DF,  p-value: < 2.2e-16
```

Can you identify the values of the regression coefficients in the output?

4.4 Categorical-by-Categorical Interactions

Factorial designs are very common in psychology, and are most often analyzed using ANOVA-based techniques, which can obscure that they are also just models.

A factorial design is one in which the predictors (IVs) are all categorical: each is a **factor** having a fixed number of **levels**. In a full-factorial design, the factors are fully crossed with each other such that each possible combination of factors is represented. We call each unique combination a **cell** of the design. You will often hear designs referred to as "a two-by-two design" (2x2), which means that there are two factors, each of which has three levels. A "three-by-three" (3x3) design is one where there are two factors, each with three levels; a "two-by-two-by-two" 2x2x2 design is one in which there are three factors, each with two levels; and so on.

Typically, factorial designs are given a tabular representation, showing all the combinations of factor levels. Below is a tabular representation of a 2x2 design.

	B_1	B_2
A_1	AB_{11}	AB_{12}
A_2	AB_{21}	AB_{22}

A 3x2 design might be shown as follows.

	B_1	B_2
A_1	AB_{11}	AB_{12}
A_2	AB_{21}	AB_{22}
A_3	AB_{31}	AB_{32}

And finally, here's a 2x2x2 design.

C_1

	B_1	B_2
A_1	ABC_{111}	ABC_{121}
A_2	ABC_{211}	ABC_{221}

C_2

	B_1	B_2
A_1	ABC_{112}	ABC_{122}
A_2	ABC_{212}	ABC_{222}

Don't confuse **factors** and **levels**!

If you hear about a study that has three treatment groups (treatment A, treatment B, and control), that is not a "three-factor (three-way) design". That is a one-factor (one-way) design with a single three-level factor (treatment condition).

There is no such thing as a factor that has only one level.

You can find out how many cells a design has by multiplying the number of levels of each factor. So, a 2x3x4 design would have 24 cells in the design.

4.4.1 Effects of cognitive therapy and drug therapy on mood

Let's consider a simple factorial design and think about the types of patterns our data can show. After we get the concepts down from this concrete example, we'll map them onto the more abstract statistical terminology.

Imagine you've running a study looking at effects of two different types of therapy for depressed patients, cognitive therapy and drug therapy. Half of the participants are randomly assigned to receive Cognitive Behavioral Therapy (CBT) and the other half get some other kind of control activity. Also, you further divide your patients through random assignment into a drug therapy group, whose members receive anti-depressants, and an control group, whose members receive a placebo. After treatment (or control/placebo), you measure their mood on a scale, with higher numbers corresponding to a more positive mood.

Let's imagine that the means we obtain below are the population means, free of measurement or sampling error. We'll take a moment to consider three different possible outcomes and what they imply about how these therapies might work independently or interactively to affect mood.

Keep in mind that you will almost **never** know the true means of any population that you care about, unless you have measured all members of the population, and done so without measurement error. Below, we're talking about the hypothetical situation where you actually know the population means and can draw conclusions without any statistical tests. Any real sample means you look at will include sampling and measurement error, and any inferences you'd make would depend on the outcome of statistical tests, rather than the observed pattern of means.

Scenario A

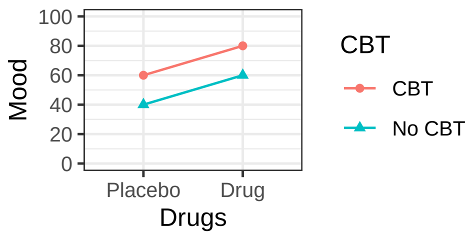

Figure 4.3: *Scenario A, plot of cell means.*

Below is a table of **cell means** and **marginal means**. The cell means are the mean values of the dependent variable (mood) at each cell of the design. The marginal means (in the margins of the table) provide the means for each row and column.

Table 4.1: *Scenario A, Table of Means.*

	No CBT	CBT	
Placebo	40	60	50
Drug	60	80	70
	50	70	

Table 4.2: *Scenario B, Table of Means.*

	No CBT	CBT	
Placebo	40	60	50
Drug	40	60	50
	40	60	

If this was our outcome, what would you conclude? Is cognitive therapy having an effect on mood? How about drug therapy. The answer to both of these questions is yes: The mean mood for people who got CBT (70; mean of column 2) is 20 points higher than the mean mood for people who didn't (50; mean of column 1).

Likewise, people who got anti-depressants showed enhanced mood (70; mean of row 2) relative to people who got the placebo (50; mean of row 1).

Now we can also ask the following question: **did the effect of cognitive therapy depend on whether or not the patient was simultaneously receiving drug therapy**? The answer to this, is no. To see why, note that for the Placebo group (Row 1), cognitive therapy increased mood by 20 points (from 40 to 60). But this was the same for the Drug group: there was an increase of 20 points from 60 to 80. So, no evidence that the effect of one factor on mood depends on the other.

Scenario B

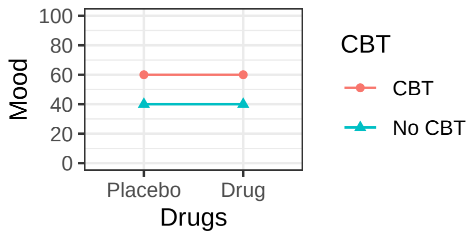

Figure 4.4: *Scenario B, plot of cell means.*

In this scenario, we also see that CBT improved mood (again, by 20 points), but there was no effect of Drug Therapy (equal marginal means of 50 for row

Table 4.3: *Scenario C, Table of Means.*

	No CBT	CBT	
Placebo	40	60	50
Drug	50	90	70
	45	75	

1 and row 2). We can also see here that the effect of CBT also didn't depend upon Drug therapy; there is an increase of 20 points in each row.

Scenario C

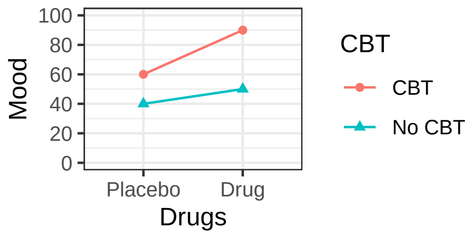

Figure 4.5: *Scenario C, plot of cell means.*

Following the logic in previous sections, we see that overall, people who got cognitive therapy showed elevated mood relative to control (75 vs 45), and that people who got drug therapy also showed elevated mood relative to placebo (70 vs 50). But there is something else going on here: it seems that the effect of cognitive therapy on mood was **more pronounced for patients who were also receiving drug therapy**. For patients on antidepressants, there was a 40 point increase in mood relative to control (from 50 to 90; row 2 of the table). For patients who got the placebo, there was only a 20 point increase in mood, from 40 to 60 (row 1 of the table). So in this hypothetical scenario, **the effect of cognitive therapy depends on whether or not there is also ongoing drug therapy.**

4.4.2 Effects in a factorial design

If you understand the basic patterns of effects described in the previous section, you are then ready to map these concepts onto statistical language.

4.4.2.1 Main effect

Main effect: The effect of a factor on the DV **ignoring** the other factors in the design.

The test of a main effect is a test of the equivalence of marginal means. So in Scenario A above, when you compared the row means for drug therapy, you were assessing the main effect of this factor on mood. The null hypothesis would be that the two marginal means are equal:

$$\bar{Y}_{1..} = \bar{Y}_{2..}$$

where $Y_{i..}$ is the mean of row i, ignoring the column factor.

If you have a factor with k levels where $k > 2$, the null hypothesis for the main effect is

$$\bar{Y}_{1..} = \bar{Y}_{2..} = ... = \bar{Y}_{k..},$$

i.e., that all of the row (or column) means are equal.

4.4.2.2 Simple effect

A **Simple effect** is the effect of one factor at a specific level of another factor (i.e., holding that factor constant at a particular value).

So for instance, in Scenario C, we talked about the effect of CBT for participants in the anti-depressant group. In that case, the simple effect of CBT for participants receiving anti-depressants was 40 units.

We could also talk about the simple effect of drug therapy for patients who received cognitive therapy. In Scenario C, this was an increase in mood from 60 to 90 (column 2).

4.4.2.3 Interaction

We say that an **interaction** is present when the effect of one variable differs across the levels of another variable.

A more mathematical definition is that an interaction is present when the simple effects of one factor differ across the levels of another factor. We saw this in Scenario C, with a 40 point boost of CBT for the anti-depressant group, compared to a boost of 20 for the placebo group. Perhaps the elevated mood caused by the anti-depressants made patients more susceptable to CBT.

The main point here is that we say there is a simple interaction between A and B when the simple effects of A differ across the levels of B. You could also check whether the simple effects of B differ across A. It is not possible for one of these statements to be true without the other also being true, so it doesn't matter which way you look at the simple effects.

4.4.3 Higher-order designs

Two-factor (also known as "two-way") designs are very common in psychology and neuroscience, but sometimes you will also see designs with more than two factors, such as a 2x2x2 design.

To figure out the number of effects we have of different kinds, we use the formula below, which gives us the number of possible combinations for n elements take k at a time:

$$\frac{n!}{k!(n-k)!}$$

Rather than actually computing this by hand, we can just use the `choose(n, k)` function in R.

For any design with n factors, you will have:

- n main effects;
- $\frac{n!}{2!(n-2)!}$ two-way interactions;
- $\frac{n!}{3!(n-3)!}$ three-way interactions;
- $\frac{n!}{4!(n-4)!}$ four-way interactions… and so forth.

So if we have a three-way design, e.g., a 2x2x2 with factors A, B, and C, we would have 3 main effects: A, B, and C. We would have `choose(3, 2)` = three two way interactions: AB, AC, and BC, and `choose(3, 3)` = one three-way interaction: ABC.

Three-way interactions are hard to interpret, but what they imply is that the **simple interaction** between any two given factors varies across the level of the third factor. For example, it would imply that the AB interaction at C_1 would be different from the AB interaction at C_2.

If you have a four way design, you have four main effects, `choose(4, 2) =6` two-way interactions, `choose(4, 3) =4` three-way interactions, and one four-way interaction. It is next to impossible to interpret results from a four-way design, so keep your designs simple!

4.5 The GLM for a factorial design

Now let's look at the math behind these models. The typically way you'll see the GLM for an ANOVA written for a 2x2 factorial design uses "ANOVA" notation, like so:

$$Y_{ijk} = \mu + A_i + B_j + AB_{ij} + S(AB)_{ijk}.$$

In the above formula,

- Y_{ijk} is the score for observation k at level i of A and level j of B;
- μ is the grand mean;
- A_i is the main effect of factor A at level i of A;
- B_j is the main effect of factor B at level j of B;
- AB_{ij} is the AB interaction at level i of A and level j of B;
- $S(AB)_{ijk}$ is the residual.

An important mathematical fact is that the individual main and interaction effects sum to zero, often written as:

- $\Sigma_i A_i = 0$;
- $\Sigma_j B_j = 0$;
- $\Sigma_{ij} AB_{ij} = 0$.

The best way to understand these effects is to see them in a decomposition table. Study the decomposition table belo wfor 12 simulated observations from a 2x2 design with factors A and B. The indexes i, j, and k are provided just to help you keep track of what observation you are dealing with. Remember that i indexes the level of factor A, j indexes the level of factor B, and k indexes the observation number within the cell AB_{ij}.

```
## # A tibble: 12 x 9
##        Y     i     j     k    mu   A_i   B_j AB_ij   err
##    <dbl> <int> <int> <int> <dbl> <dbl> <dbl> <dbl> <int>
##  1    11     1     1     1    10     4    -2    -1     0
##  2    14     1     1     2    10     4    -2    -1     3
##  3     8     1     1     3    10     4    -2    -1    -3
##  4    17     1     2     1    10     4     2     1     0
##  5    15     1     2     2    10     4     2     1    -2
##  6    19     1     2     3    10     4     2     1     2
##  7     8     2     1     1    10    -4    -2     1     3
##  8     4     2     1     2    10    -4    -2     1    -1
##  9     3     2     1     3    10    -4    -2     1    -2
## 10    10     2     2     1    10    -4     2    -1     3
## 11     7     2     2     2    10    -4     2    -1     0
## 12     4     2     2     3    10    -4     2    -1    -3
```

4.5.1 Estimation equations

These are the equations you would used to estimate effects in an ANOVA.

- $\hat{\mu} = Y_{...}$
- $\hat{A}_i = Y_{i..} - \hat{\mu}$
- $\hat{B}_j = Y_{.j.} - \hat{\mu}$
- $\widehat{AB}_{ij} = Y_{ij.} - \hat{\mu} - \hat{A}_i - \hat{B}_j$

Note that the Y variable with the dots in the subscripts are means of Y, taken while ignoring anything appearing as a dot. So $Y_{...}$ is mean of Y, $Y_{i..}$ is the mean of Y at level i of A, $Y_{.j.}$ is the mean of Y at level j of B, and $Y_{ij.}$ is the mean of Y at level i of A and level j of B, i.e., the cell mean ij.

4.5.2 Factorial App

Launch this web application and experiment with factorial designs until you understand the key concepts of main effects and interactions in a factorial design.

4.6 Code your own categorical predictors in factorial designs

Many studies in psychology—especially experimental psychology—involve categorical independent variables. Analyzing data from these studies requires care in specifying the predictors, because the defaults in R are not ideal for experimental situations.

Let's say you have 2x2 designed experiment with factors priming condition (priming vs. no priming) and linguistic structure (noun vs verb). These columns can be represented as type `character` or `factor`; in the latter case, they are implicitly converted to type `factor` before fitting the model, and then R will apply the default numerical coding for factors, which is 'treatment' (0, 1) coding.

If you're used to running ANOVAs, the results that you get from fitting a linear model will *not* match ANOVA output, as we'll see below. That is because you need to use a different coding scheme to get ANOVA-like output.

First, let's define our little data set, `dat`.

```
## demo for why you should avoid factors
dat <- tibble(
  subject = factor(1:16),
  priming = rep(c("yes", "no"), each = 8),
  structure = rep(rep(c("noun", "verb"), each = 4), 2),
  RT = rnorm(16, 800, 20))

dat
```

```
## # A tibble: 16 x 4
##    subject priming structure    RT
##    <fct>   <chr>   <chr>     <dbl>
##  1 1       yes     noun       792.
##  2 2       yes     noun       771.
##  3 3       yes     noun       760.
##  4 4       yes     noun       811.
##  5 5       yes     verb       787.
##  6 6       yes     verb       820.
##  7 7       yes     verb       768.
##  8 8       yes     verb       806.
##  9 9       no      noun       791.
## 10 10      no      noun       824.
## 11 11      no      noun       817.
## 12 12      no      noun       790.
## 13 13      no      verb       790.
## 14 14      no      verb       813.
## 15 15      no      verb       810.
## 16 16      no      verb       763.
```

This is between subjects data, so we can fit a model using `lm()`. In the model, we include effects of `priming` and `structure` as well as their interaction. Instead of typing `priming + structure + priming:structure` we can simply type the shortcut `priming * structure`.

```
ps_mod <- lm(RT ~ priming * structure, dat)

summary(ps_mod)
```

```
##
## Call:
## lm(formula = RT ~ priming * structure, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -30.782 -14.711   2.036  16.466  27.755
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)                805.57      10.83  74.409   <2e-16 ***
## primingyes                 -22.08      15.31  -1.442    0.175
## structureverb              -11.49      15.31  -0.750    0.468
## primingyes:structureverb    23.20      21.65   1.071    0.305
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 21.65 on 12 degrees of freedom
## Multiple R-squared:  0.1481,Adjusted R-squared:  -0.06493
## F-statistic: 0.6951 on 3 and 12 DF,  p-value: 0.5726
```

Note that in the output the predictors are shown as `primingyes` and `structureverb`. The value `yes` is a level of `priming`; the level **not shown** is the one chosen as baseline, and in the default treatment coding scheme, the not-shown level (`no`) is coded as 0, and the shown level (`yes`) is coded as 1. Likewise, for `structure`, `noun` is coded as 0 and `verb` is coded as 1.

This is not ideal, for reasons we will discuss further below. But I want to show you a further quirk of using factor variables as predictors.

Let's say we wanted to test the effect of `priming` by itself using model comparison. To do this, we would fit another model where we exclude this effect while keeping the interaction. Despite what you may have heard to the contrary, in a fully randomized, balanced experiment, all factors are orthogonal, and so it is completely legitimate to drop a main effect while leaving an interaction term in the model.

```
ps_mod_nopriming <- lm(RT ~ structure + priming:structure, dat)
```

OK, now that we've dropped `priming`, we should have 3 parameter estimates instead of 4. Let's check.

```
## not right!
coef(ps_mod_nopriming)
```

```
##               (Intercept)             structureverb structurenoun:primingyes 
##                805.566187                -11.485457               -22.081224 
## structureverb:primingyes 
##                  1.117333
```

There are still 4 of them, and we're suddenly getting `primingyes:structureverb`. This is weird and *not at all* what we intended. If we try to do the model comparison:

```
## nonsense result
anova(ps_mod_nopriming, ps_mod)
```

```
## Analysis of Variance Table
## 
## Model 1: RT ~ structure + priming:structure
## Model 2: RT ~ priming * structure
##   Res.Df  RSS Df  Sum of Sq F Pr(>F)
## 1     12 5626                       
## 2     12 5626  0 9.0949e-13
```

we'd get nonsensical results.

Is this a bug? No. It was a (in my view, heavy handed) design choice by the R creators to try to prevent everyone from doing something that at least some of us should be able to do at least some of the time.

But we can do whatever we please if instead of using factors we define our own numerical predictors. This adds a bit of work but avoids other headaches and mistakes that we might make by using factors. Also, being very explicit about how predictors are defined is probably a good thing.

You'll sometimes need `factor` variables. I often use them to get things to plot in the right way using `ggplot2`, or when I need to tabulating observations and there are some combinations with zero counts. But I recommend against using `factors` in statistical models, especially if your model includes interactions. Use numerical predictors instead.

4.7 Coding schemes for categorical variables

Many experimentalists who are trying to make the leap from ANOVA to linear mixed-effects models (LMEMs) in R struggle with the coding of categorical predictors. It is unexpectedly complicated, and the defaults provided in R turn out to be wholly inappropriate for factorial experiments. Indeed, using those defaults with factorial experiments can lead researchers to draw erroneous conclusions from their data.

To keep things simple, we'll start with situations where design factors have no more than two levels before moving on to designs with more than three levels.

4.7.1 Simple versus main effects

It is important that you understand the difference between a **simple effect** and a **main effect**, and between a **simple interaction** and a **main interaction** in a three-way design.

In an $A \times B$ design, the simple effect of A is the effect of A **controlling** for B, while the main effect of A is the effect of A **ignoring** B. Another way of looking at this is to consider the cell means ($\bar{Y}_{11}$, $\bar{Y}_{12}$, $\bar{Y}_{21}$, and $\bar{Y}_{22}$) and marginal means ($\bar{Y}_{1.}$, $\bar{Y}_{2.}$, $\bar{Y}_{.1}$, and $\bar{Y}_{.2}$) in a factorial design. (The dot subscript tells you to "ignore" the dimension containing the dot; e.g., $\bar{Y}_{.1}$ tells you to take the mean of the first column ignoring the row variable.) To test the main effect of A is to test the null hypothesis that $\bar{Y}_{1.} = \bar{Y}_{2.}$. To test a simple effect of A—the effect of A at a particular level of B—would be, for instance, to test the null hypothesis that $\bar{Y}_{11} = \bar{Y}_{21}$.

	\(B_1\)	\(B_2\)		
\(A_1\)	$\bar{Y}_{11}$	$\bar{Y}_{12}$		$\bar{Y}_{1.}$
\(A_2\)	$\bar{Y}_{21}$	$\bar{Y}_{22}$		$\bar{Y}_{2.}$
	$\bar{Y}_{.1}$	$\bar{Y}_{.2}$		

The distinction between **simple interactions** and **main interactions** has the same logic: the simple interaction of AB in an ABC design is the interaction of AB at a particular level of C; the main interaction of AB is the interaction **ignoring** C. The latter is what we are usually talking about when we talk about lower-order interactions in a three-way design. It is also what we are given in the output from standard ANOVA procedures, e.g., the `aov()` function in R, SPSS, SAS, etc.

4.7.2 The key coding schemes

Generally, the choice of a coding scheme impacts the interpretation of:

1. the intercept term; and
2. the interpretation of the tests for all but the highest-order effects and interactions in a factorial design.

It also can influence the interpretation/estimation of random effects in a mixed-effects model (see this blog post for further discussion). If you have a design with only a single two-level factor, and are using a maximal random-effects structure, the choice of coding scheme doesn't really matter.

There are many possible coding schemes (see `?contr.treatment` for more information). The most relevant ones are **treatment**, **sum**, and **deviation**. Sum and deviation coding can be seen as special cases of **effect** coding; by effect coding, people generally mean codes that sum to zero.

For a two-level factor, you would use the following codes:

Scheme	\(A_1\)	\(A_2\)
Treatment (dummy)	\(0\)	\(1\)
Sum	\(-1\)	\(1\)
Deviation	\(-\frac{1}{2}\)	\(\frac{1}{2}\)

The default in R is to use treatment coding for any variable defined as a =factor= in the model (see
lm(Y ~ A * B * C)

The figure below spells out the notation for the various cell and marginal means for a 2x2x2 design.

C_1

	\(B_1\)	\(B_2\)		
\(A_1\)	$\bar{Y}_{111}$	$\bar{Y}_{121}$		$\bar{Y}_{1.1}$
\(A_2\)	$\bar{Y}_{211}$	$\bar{Y}_{221}$		$\bar{Y}_{2.1}$
	$\bar{Y}_{.11}$	$\bar{Y}_{.21}$		

C_2

	\(B_1\)	\(B_2\)		
\(A_1\)	$\bar{Y}_{112}$	$\bar{Y}_{122}$		$\bar{Y}_{1.2}$
\(A_2\)	$\bar{Y}_{212}$	$\bar{Y}_{222}$		$\bar{Y}_{2.2}$
	$\bar{Y}_{.12}$	$\bar{Y}_{.22}$		

The table below provides the interpretation for various effects in the model under the three different coding schemes. Note that Y is the dependent variable, and the dots in the subscript mean to "ignore" the corresponding dimension. Thus, $\bar{Y}_{.1.}$ is the mean of B_1 (ignoring factors A and C) and $\bar{Y}_{...}$ is the "grand mean" (ignoring all factors).

term	treatment	
\(\mu\)	$\bar{Y}_{111}$	
\(A\)	\(\bar{Y}_{211} - \bar{Y}_{111}\)	
\(B\)	\(\bar{Y}_{121} - \bar{Y}_{111}\)	
\(C\)	\(\bar{Y}_{112} - \bar{Y}_{111}\)	
\(AB\)	\((\bar{Y}_{221} - \bar{Y}_{121}) - (\bar{Y}_{211} - \bar{Y}_{111})\)	\(\frac{(\
\(AC\)	\((\bar{Y}_{212} - \bar{Y}_{211}) - (\bar{Y}_{112} - \bar{Y}_{111})\)	\(\frac{(\
\(BC\)	\((\bar{Y}_{122} - \bar{Y}_{112}) - (\bar{Y}_{121} - \bar{Y}_{111})\)	\(\frac{(\

For the three way $A \times B \times C$ interaction:

scheme	
treatment	\(\displaystyle\left[\displaystyle\left(\bar{Y}_{221} - \bar{Y}_{121}\right) - \dis
sum	\(\frac{\displaystyle\left[\displaystyle\left(\bar{Y}_{221} - \bar{Y}_{121}\right) - \d
deviation	\(\displaystyle\left[\displaystyle\left(\bar{Y}_{221} - \bar{Y}_{121}\right) - \dis

Note that the inferential tests of $A \times B \times C$ will all have the same outcome, despite the parameter estimate for sum coding being one-eighth of that for the other schemes. For all lower-order effects, sum and deviation coding will give different parameter estimates but identical inferential outcomes. Both of these schemes provide identical tests of the canonical main effects and main interactions for a three-way ANOVA. In contrast, treatment (dummy) coding will provide inferential tests of simple effects and simple interactions. So, if what you are interested in getting are the "canonical" tests from ANOVA, use sum or deviation coding.

4.7.3 What about factors with more than two levels?

A factor with k levels requires $k - 1$ variables. Each predictor contrasts a particular "target" level of the factor with a level that you (arbitrarily) choose as the "baseline" level. For instance, for a three-level factor A with $A1$ chosen as the baseline, you'd have two predictor variables, one of which compares $A2$ to $A1$ and the other of which compares $A3$ to $A1$.

For treatment (dummy) coding, the target level is set to 1, otherwise 0.

For sum coding, the levels must sum to zero, so for a given predictor, the target level is given the value 1, the baseline level is given the value -1, and any other level is given the value 0.

For deviation coding, the values must also sum to 0. Deviation coding is recommended whenever you are trying to draw ANOVA-style inferences. Under this scheme, the target level gets the value $\frac{k-1}{k}$ while any non-target level gets the value $-\frac{1}{k}$.

Fun fact: Mean-centering treatment codes (on balanced data) will give you deviation codes.

4.7.4 Example: Three-level factor

4.7.4.1 Treatment (Dummy)

level	A2v1	A3v1
A1	0	0
A2	1	0
A3	0	1

4.7.4.2 Sum

level	A2v1	A3v1
A1	-1	-1
A2	1	0
A3	0	1

4.7.4.3 Deviation

level	A2v1	A3v1
A1	\(-\frac{1}{3}\)	\(-\frac{1}{3}\)
A2	\(\frac{2}{3}\)	\(-\frac{1}{3}\)
A3	\(-\frac{1}{3}\)	\(\frac{2}{3}\)

4.7.4.4 Example: Five-level factor

4.7.4.5 Treatment (Dummy)

level	A2v1	A3v1	A4v1	A5v1
A1	0	0	0	0
A2	1	0	0	0
A3	0	1	0	0
A4	0	0	1	0
A5	0	0	0	1

4.7.4.6 Sum

level	A2v1	A3v1	A4v1	A5v1
A1	-1	-1	-1	-1
A2	1	0	0	0
A3	0	1	0	0
A4	0	0	1	0
A5	0	0	0	1

4.7.4.7 Deviation

level	A2v1	A3v1	A4v1	A5v1
A1	\(-\frac{1}{5}\)	\(-\frac{1}{5}\)	\(-\frac{1}{5}\)	\(-\frac{1}{5}\)
A2	\(\frac{4}{5}\)	\(-\frac{1}{5}\)	\(-\frac{1}{5}\)	\(-\frac{1}{5}\)
A3	\(-\frac{1}{5}\)	\(\frac{4}{5}\)	\(-\frac{1}{5}\)	\(-\frac{1}{5}\)
A4	\(-\frac{1}{5}\)	\(-\frac{1}{5}\)	\(\frac{4}{5}\)	\(-\frac{1}{5}\)
A5	\(-\frac{1}{5}\)	\(-\frac{1}{5}\)	\(-\frac{1}{5}\)	\(\frac{4}{5}\)

4.7.5 How to create your own numeric predictors

Let's assume that your data is contained in a table `dat` like the one below.

```r
## create your own numeric predictors
## make an example table
dat <- tibble(Y = rnorm(12),
              A = rep(paste0("A", 1:3), each = 4))
```

Click to view example data

Y	A
-0.75	A1
0.03	A1
-0.22	A1
1.09	A1
-0.42	A2
-0.68	A2
2.09	A2
-1.37	A2
0.27	A3
0.47	A3
-0.95	A3
0.16	A3

4.7.5.1 The `mutate()` / `if_else()` / `case_when()` approach for a three-level factor

4.7.5.2 Treatment

```r
## examples of three level factors
## treatment coding
dat_treat <- dat %>%
  mutate(A2v1 = if_else(A == "A2", 1L, 0L),
         A3v1 = if_else(A == "A3", 1L, 0L))
```

Click to view resulting table

```
## # A tibble: 12 x 4
##          Y A      A2v1  A3v1
##      <dbl> <chr> <int> <int>
##  1 -0.752  A1        0     0
##  2  0.0251 A1        0     0
##  3 -0.218  A1        0     0
##  4  1.09   A1        0     0
##  5 -0.417  A2        1     0
##  6 -0.683  A2        1     0
##  7  2.09   A2        1     0
##  8 -1.37   A2        1     0
##  9  0.268  A3        0     1
## 10  0.472  A3        0     1
## 11 -0.946  A3        0     1
## 12  0.155  A3        0     1
```

4.7.5.3 Sum

```
## sum coding
dat_sum <- dat %>%
  mutate(A2v1 = case_when(A == "A1" ~ -1L, # baseline
                          A == "A2" ~ 1L,  # target
                          TRUE      ~ 0L), # anything else
         A3v1 = case_when(A == "A1" ~ -1L, # baseline
                          A == "A3" ~  1L, # target
                          TRUE      ~ 0L)) # anything else
```

Click to view resulting table

```
## # A tibble: 12 x 4
##          Y A      A2v1  A3v1
##      <dbl> <chr> <int> <int>
##  1 -0.752  A1       -1    -1
##  2  0.0251 A1       -1    -1
##  3 -0.218  A1       -1    -1
##  4  1.09   A1       -1    -1
##  5 -0.417  A2        1     0
##  6 -0.683  A2        1     0
##  7  2.09   A2        1     0
##  8 -1.37   A2        1     0
##  9  0.268  A3        0     1
## 10  0.472  A3        0     1
## 11 -0.946  A3        0     1
## 12  0.155  A3        0     1
```

4.7.5.4 Deviation

```
## deviation coding
## baseline A1
dat_dev <- dat %>%
  mutate(A2v1 = if_else(A == "A2", 2/3, -1/3), # target A2
         A3v1 = if_else(A == "A3", 2/3, -1/3)) # target A3
```

Click to view resulting table

```
dat_dev
```

```
## # A tibble: 12 x 4
##          Y A       A2v1   A3v1
##      <dbl> <chr>  <dbl>  <dbl>
##  1 -0.752  A1    -0.333 -0.333
##  2  0.0251 A1    -0.333 -0.333
##  3 -0.218  A1    -0.333 -0.333
##  4  1.09   A1    -0.333 -0.333
##  5 -0.417  A2     0.667 -0.333
##  6 -0.683  A2     0.667 -0.333
##  7  2.09   A2     0.667 -0.333
##  8 -1.37   A2     0.667 -0.333
##  9  0.268  A3    -0.333  0.667
## 10  0.472  A3    -0.333  0.667
## 11 -0.946  A3    -0.333  0.667
## 12  0.155  A3    -0.333  0.667
```

4.7.6 Conclusion

The interpretation of all but the highest order effect depends on the coding scheme.

With treatment coding, you are looking at **simple** effects and **simple** interactions, not **main** effects and **main** interactions.

The parameter estimates for sum coding differs from deviation coding only in the magnitude of the parameter estimates, but have identical interpretations.

Because it is not subject to the scaling effects seen under sum coding, deviation should be used by default for ANOVA-style designs.

The default coding scheme for factors is R is "treatment" coding.

So, anytime you declare a variable as type `factor` and use this variable as a predictor in your regression model, R will automatically create treatment-coded variables.

Take-home message: when analyzing factorial designs in R using regression, to obtain the canonical ANOVA-style interpretations of main effects and interactions use deviation coding and NOT the default treatment coding.

Chapter 5

Introducing Linear Mixed-Effects Models

5.1 Learning objectives

- express multi-level designs in mathematical format (DGP)
- analyze and interpret parameter estimates using `lme4::lmer()`
- simulate data from a multi-level design with a single predictor

In the exercises below, we'll be working with the `sleepstudy` data, a built in dataset from the lme4 package. To get information about the data, after loading in the lme4 package, type `?sleepstudy` at the prompt.

5.2 Modeling multi-level data

The `sleepstudy` data is included as a built-in dataset in the `lme4` package for R (Bates et al., 2015). Some of the ideas presented in this section come from McElreath (2020) Statistical Rethinking textbook as well as from Tristan Mahr's blog post on partial pooling.

Let's start by looking at the documentation for the `sleepstudy` dataset. After loading the **`lme4`** package, you can access the documentation by typing `?sleepstudy` in the console.

```
sleepstudy                package:lme4                 R Documentation

Reaction times in a sleep deprivation study

Description:
     The average reaction time per day for subjects in a sleep
     deprivation study. On day 0 the subjects had their normal amount
     of sleep.  Starting that night they were restricted to 3 hours of
     sleep per night.  The observations represent the average reaction
     time on a series of tests given each day to each subject.

Format:

     A data frame with 180 observations on the following 3 variables.

     'Reaction' Average reaction time (ms)

     'Days' Number of days of sleep deprivation

     'Subject' Subject number on which the observation was made.

Details:

     These data are from the study described in Belenky et al.  (2003),
     for the sleep-deprived group and for the first 10 days of the
     study, up to the recovery period.

References:

     Gregory Belenky, Nancy J. Wesensten, David R. Thorne, Maria L.
     Thomas, Helen C. Sing, Daniel P. Redmond, Michael B. Russo and
     Thomas J. Balkin (2003) Patterns of performance degradation and
     restoration during sleep restriction and subsequent recovery: a
     sleep dose-response study. _Journal of Sleep Research_ *12*, 1-12.
```

Note that these data meet our definition of multilevel data due to multistage sampling: Participants are sampled from a population, and then their mean reaction times are measured on each of the 10 days in the study. Even though we don't have the trial-by-trial information used to calculate the means, this is still multilevel data, because of the multiple observations on the response variable (reaction time) for each subject.

Multi-level data of this type is extremely common in psychology. However, if you open up most introductory statistics textbooks and read through to the end, you'll learn about t-tests, repeated-measures and mixed-model ANOVA, and multiple regression, but none of the techniques that you learn would be applicable to multi-level data of this sort. It suggests a serious gap in training, one which should be filled by teaching multi-level modeling. All of the other techniques—t-test, ANOVA, multiple regression—can be seen as special cases of the more general multi-level modeling approach. My view is that multi-level modeling should be taught as the **default**, not as an advanced approach that is too far beyond the reach of undergraduates.

Let's consider the `sleepstudy` data. The dataset contains eighteen participants from the three-hour sleep condition. Each day, over 10 days, participants performed a ten-minute "psychomotor vigilance test" where they had to monitor a display and press a button as quickly as possible each time a stimulus appeared. The dependent measure in the dataset is the participant's average response time (RT) on the task for that day.

It is always good to start by looking at the data. Here is data for a single subject.

```r
library("lme4")
library("tidyverse")

just_308 <- sleepstudy %>%
  filter(Subject == "308")

ggplot(just_308, aes(x = Days, y = Reaction)) +
  geom_point() +
  scale_x_continuous(breaks = 0:9)
```

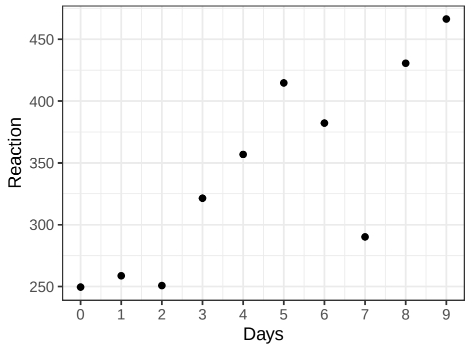

Figure 5.1: *Data from a single subject in Belenky et al. (2003)*

It looks like RT is increasing with each additional day of sleep deprivation, at least from day 2 to day 10.

Exercise

Use ggplot to recreate the plot below, which shows data for all 18 subjects.

Hint, please

Just above, you were given the code to make a plot for a single participant. Adapt this code to show all of the participants by getting rid of the `filter()` statement and adding a *ggplot2* function that starts with `facet_`.

Show me

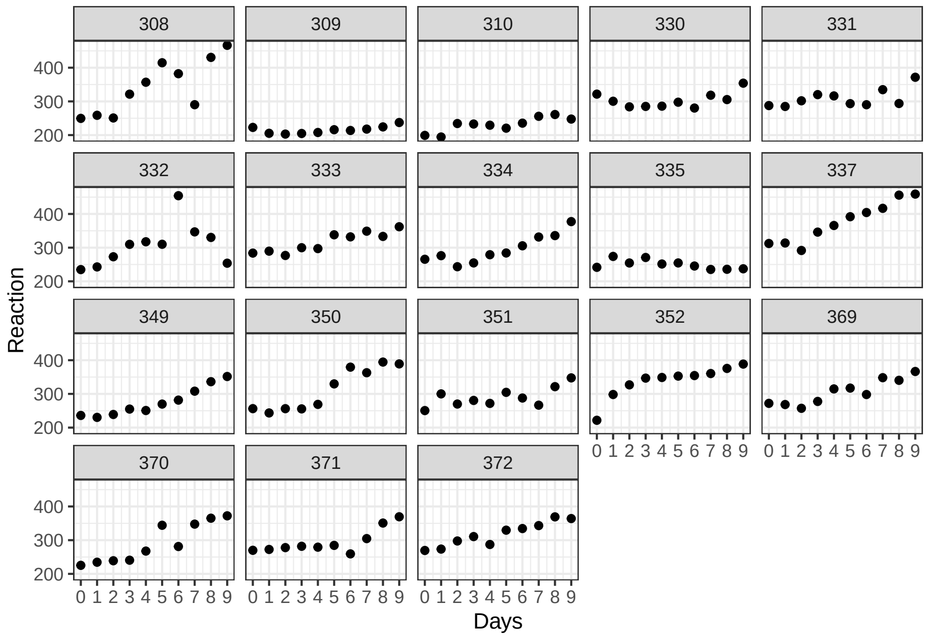

Figure 5.2: *Data from Belenky et al. (2003)*

Same as above, except you just add one line: `facet_wrap(~Subject)`

```
ggplot(sleepstudy, aes(x = Days, y = Reaction)) +
  geom_point() +
  scale_x_continuous(breaks = 0:9) +
  facet_wrap(~Subject)
```

5.3 How to model this data?

Before proceeding, there is one important thing to note. Unfortunately, the description of the `sleepstudy` data in the `lme4` package is inaccurate. The documentation states:

> On day 0 the subjects had their normal amount of sleep. Starting that night they were restricted to 3 hours of sleep per night.

However, this is what Belenky et al. (2003) actually say (p. 2 of the PDF):

> The first 3 days (T1, T2 and B) were adaptation and training (T1 and T2) and baseline (B) and subjects were required to be in bed from 23:00 to 07:00 h [8 h required time in bed (TIB)]. On the third day (B), baseline measures were taken. Beginning on the fourth day and continuing for a total of 7 days (E1–E7) subjects were in

> one of four sleep conditions [9 h required TIB (22:00–07:00 h), 7 h required TIB (24:00–07:00 h), 5 h required TIB (02:00–07:00 h), or 3 h required TIB (04:00–07:00 h)], effectively one sleep augmentation condition, and three sleep restriction conditions.

One thing to note is that the dataset includes data from only one of the three conditions (the 3h TIB condition). But the more important thing is that the documentation of `sleepstudy` is wrong: there were seven nights of sleep restriction, not nine, with the first night of restriction occurring after the third day. The first two days, coded as `0`, `1`, were adaptation and training. The day coded as `2`, where the baseline measurement was taken should be the place where we start our analysis. If we include the days `0` and `1` in our analysis, this might bias our results, since any changes in performance during the first two days have to do with training, not sleep restriction.

Exercise

Remove from the dataset observations where `Days` is coded `0` or `1`, and then make a new variable `days_deprived` from the `Days` variable so that the sequence starts at day 2, with day 2 being re-coded as day 0, day 3 as day 1, day 4 as day 2, etc. This new variable now tracks the number of days of sleep deprivation. Store the new table as `sleep2`.

Solution

```r
sleep2 <- sleepstudy %>%
  filter(Days >= 2L) %>%
  mutate(days_deprived = Days - 2L)
```

It is always a good idea to double check that the code works as intended. First, look at it:

```r
head(sleep2)
```

```
##   Reaction Days Subject days_deprived
## 1 250.8006    2     308             0
## 2 321.4398    3     308             1
## 3 356.8519    4     308             2
## 4 414.6901    5     308             3
## 5 382.2038    6     308             4
## 6 290.1486    7     308             5
```

And check that `Days` and `days_deprived` match up.

```r
sleep2 %>%
  count(days_deprived, Days)
```

```
##   days_deprived Days  n
## 1             0    2 18
## 2              1     3 18
## 3              2     4 18
## 4              3     5 18
## 5              4     6 18
## 6              5     7 18
## 7              6     8 18
## 8              7     9 18
```

Looks good. Note that the variable `n` in generated by `count()` and tells you how many rows there are for each unique combination of `Days` and `days_deprived`. In this case, there were 18, one row for each participant.

Now let's re-plot the data looking at just these eight data points from Day 0 to Day 7. We've just copied the code from above, substituting `sleep2` for `sleepstudy` and using `days_deprived` for our `x` variable.

```
ggplot(sleep2, aes(x = days_deprived, y = Reaction)) +
  geom_point() +
  scale_x_continuous(breaks = 0:7) +
  facet_wrap(~Subject) +
  labs(y = "Reaction Time", x = "Days deprived of sleep (0 = baseline)")
```

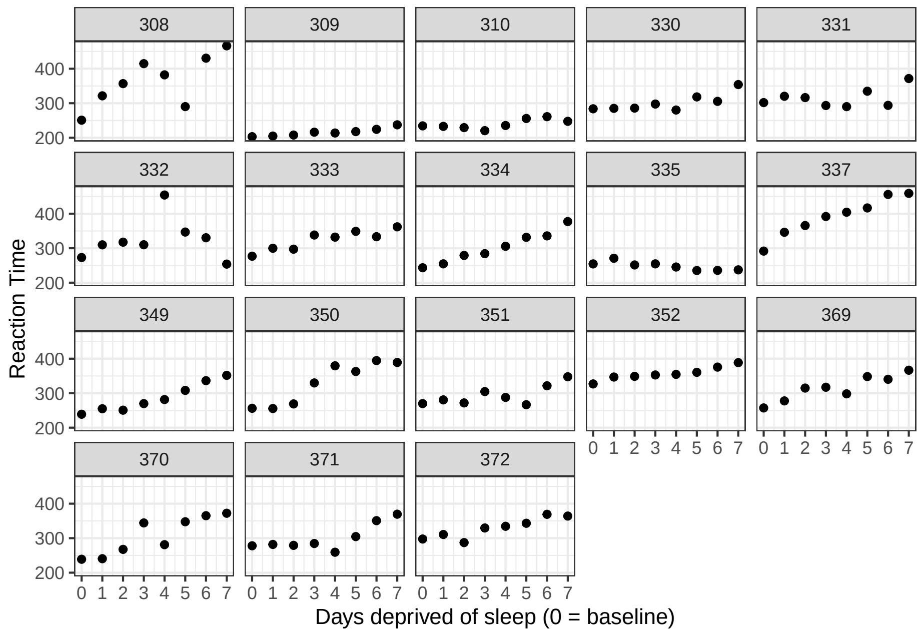

Figure 5.3: *Data from Belenky et al. (2003), showing reaction time at baseline (0) and after each day of sleep deprivation.*

Take a moment to think about how me might model the relationship between `days_deprived` and `Reaction`. Does reaction time increase or decrease with increasing sleep deprivation? Is the relationship roughly stable or does it change with time?

With only one exception (subject 335) it looks like reaction time increases with each additional day of sleep deprivation. It looks like we could fit a line to each participant's data. Recall the general equation for a line is of the form **y = y-intercept + slope × x**. In regression, the we usually express a linear relationship with the formula

$$Y = \beta_0 + \beta_1 X$$

where β_0 is the y-intercept and β_1 is the slope, parameters whose values we estimate from the data.

The lines will all differ in intercept (mean RT at day zero, before the sleep deprivation began) and slope (the change in RT with each additional day of sleep deprivation). But should we fit the same line to everyone? Or a totally different line for each subject? Or something somewhere in between?

Let's start by considering three different approaches we might take. Following McElreath, we will distinguish these approaches by calling them **complete pooling**, **no pooling**, and **partial pooling**.

5.3.1 Complete pooling: One size fits all

The **complete pooling** approach is a "one-size-fits-all" model: it estimates a single intercept and slope for the entire dataset, ignoring the fact that different subjects might vary in their intercepts or slopes. If that sounds like a bad approach, it is; but you know this because you've already visualized the data and noted that the pattern for each participant would seem to require different y-intercept and slope values.

Fitting one line is called the "complete pooling" approach because we pool together data from all subjects to get single estimates for an overall intercept and slope. The GLM for this approach is simply

$$Y_{sd} = \beta_0 + \beta_1 X_{sd} + e_{sd}$$

$$e_{sd} \sim N\left(0, \sigma^2\right)$$

where Y_{sd} is the mean RT for subject s on day d, X_{sd} is the value of `Day` associated with that case (0-7), and e_{sd} is the error.

We would fit such a model in R using the `lm()` function, e.g.:

```r
cp_model <- lm(Reaction ~ days_deprived, sleep2)

summary(cp_model)
```

```
##
## Call:
## lm(formula = Reaction ~ days_deprived, data = sleep2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -112.284  -26.732    2.143   27.734  140.453
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)    267.967      7.737  34.633  < 2e-16 ***
## days_deprived   11.435      1.850   6.183 6.32e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 50.85 on 142 degrees of freedom
## Multiple R-squared:  0.2121,Adjusted R-squared:  0.2066
## F-statistic: 38.23 on 1 and 142 DF,  p-value: 6.316e-09
```

According to this model, the predicted mean response time on Day 0 is about 268 milliseconds, with an increase of about 11 milliseconds per day of deprivation, on average. We can't trust the standard errors for our regression coefficients, however, because we are assuming that all of the observations are independent (technically, that the residuals are). However, we can be pretty sure this is a bad assumption.

Let's add the model predictions to the graph that we created above. We can use `geom_abline()` to do so, specifying the intercept and slope for the line using the regression coefficients from the model fit, `coef(cp_model)`, which returns a two-element vector with intercept and slope, respectively.

```r
coef(cp_model)
```

```
##   (Intercept) days_deprived
##     267.96742      11.43543
```

```r
ggplot(sleep2, aes(x = days_deprived, y = Reaction)) +
  geom_abline(intercept = coef(cp_model)[1],
              slope = coef(cp_model)[2],
              color = 'blue') +
  geom_point() +
  scale_x_continuous(breaks = 0:7) +
  facet_wrap(~Subject) +
  labs(y = "Reaction Time", x = "Days deprived of sleep (0 = baseline)")
```

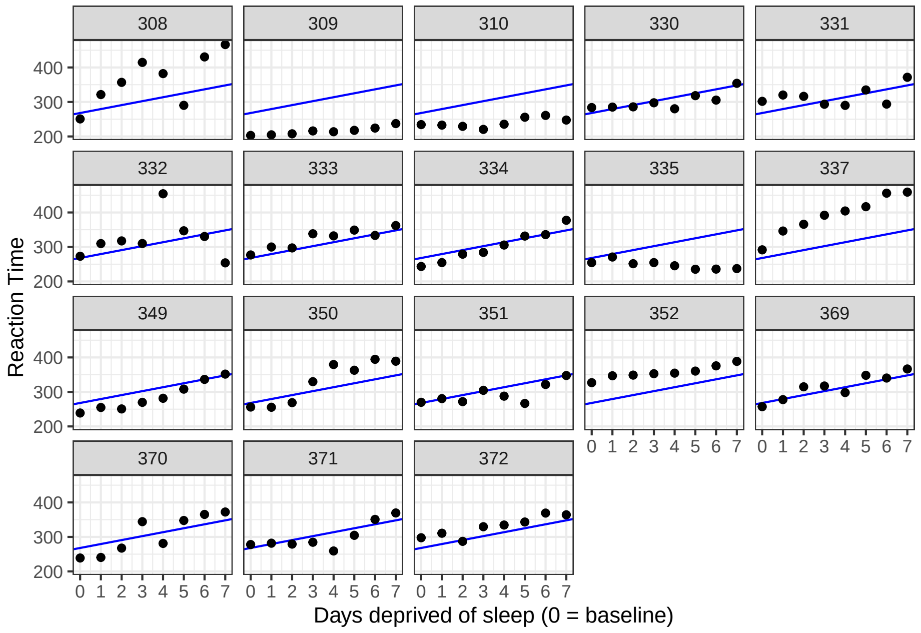

Figure 5.4: Data plotted against predictions from the complete pooling model.

The model fits the data badly. We need a different approach.

5.3.2 No pooling

Pooling all the information to get just one intercept and one slope estimate is inappropriate. Another approach would be to fit separate lines for each participant. This means that the estimates for each participant will be completely uninformed by the estimates for the other participants. In other words, we can separately estimate 18 individual intercept/slope pairs.

This model could be implemented in two ways: (1) by running separate regressions for each participant or (2) by running fixed-effects regression. We'll do the latter, so that everything is in one big model. We know how to do this already: we add in dummy codes for the `Subject` factor. We have 18 levels of this factor, so we'd need 17 dummy codes. Fortunately, R saves us from the trouble of creating the 17 variables we would need by hand. All we need to do is include `Subject` as a predictor in the model, and interact this categorical predictor with `days_deprived` to allow intercepts and slopes to vary.

The variable `Subject` in the `sleep2` dataset is nominal. We just use numbers as labels to preserve anonymity, without intending to imply that Subject 310 is one point better than Subject 309 and two points better than 308. Make sure that you define it as a factor so that it is not included as a continuous variable!

We can test whether something is a factor in various ways. One is to use `summary()` on the table.

```
sleep2 %>% summary()
```

```
##     Reaction          Days          Subject   days_deprived
##  Min.   :203.0   Min.   :2.00   308    : 8   Min.   :0.00  
##  1st Qu.:265.2   1st Qu.:3.75   309    : 8   1st Qu.:1.75  
##  Median :303.2   Median :5.50   310    : 8   Median :3.50  
##  Mean   :308.0   Mean   :5.50   330    : 8   Mean   :3.50  
##  3rd Qu.:347.7   3rd Qu.:7.25   331    : 8   3rd Qu.:5.25  
##  Max.   :466.4   Max.   :9.00   332    : 8   Max.   :7.00  
##                                 (Other):96
```

Here you can see that it is not treated as a number because rather than giving you distributional information (means, etc.) it tells you how many observations there are at each level.

You can also test it directly:

```
sleep2 %>% pull(Subject) %>% is.factor()
```

```
## [1] TRUE
```

If something is not a factor, you can make it one be re-defining it using the `factor()` function.

```
np_model <- lm(Reaction ~ days_deprived + Subject + days_deprived:Subject,
               data = sleep2)

summary(np_model)
```

```
## 
## Call:
## lm(formula = Reaction ~ days_deprived + Subject + days_deprived:Subject,
##     data = sleep2)
## 
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -106.521   -8.541    1.143    8.889  128.545 
## 
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)    
## (Intercept)              288.2175    16.4772  17.492  < 2e-16 ***
## days_deprived             21.6905     3.9388   5.507 2.49e-07 ***
## Subject309               -87.9262    23.3023  -3.773 0.000264 ***
## Subject310               -62.2856    23.3023  -2.673 0.008685 **
## Subject330              -14.9533    23.3023  -0.642 0.522422
## Subject331                9.9658    23.3023   0.428 0.669740
## Subject332               27.8157    23.3023   1.194 0.235215
## Subject333               -2.7581    23.3023  -0.118 0.906000
## Subject334              -50.2051    23.3023  -2.155 0.033422 *
## Subject335              -25.3429    23.3023  -1.088 0.279207
## Subject337               24.6143    23.3023   1.056 0.293187
## Subject349              -59.2183    23.3023  -2.541 0.012464 *
## Subject350              -40.2023    23.3023  -1.725 0.087343 .
## Subject351              -24.2467    23.3023  -1.041 0.300419
## Subject352               43.0655    23.3023   1.848 0.067321 .
## Subject369              -21.5040    23.3023  -0.923 0.358154
## Subject370              -53.3072    23.3023  -2.288 0.024107 *
## Subject371              -30.4896    23.3023  -1.308 0.193504
## Subject372                2.4772    23.3023   0.106 0.915535
## days_deprived:Subject309 -17.3334     5.5703  -3.112 0.002380 **
## days_deprived:Subject310 -17.7915     5.5703  -3.194 0.001839 **
## days_deprived:Subject330 -13.6849     5.5703  -2.457 0.015613 *
## days_deprived:Subject331 -16.8231     5.5703  -3.020 0.003154 **
## days_deprived:Subject332 -19.2947     5.5703  -3.464 0.000765 ***
## days_deprived:Subject333 -10.8151     5.5703  -1.942 0.054796 .
## days_deprived:Subject334  -3.5745     5.5703  -0.642 0.522423
## days_deprived:Subject335 -25.8995     5.5703  -4.650 9.47e-06 ***
## days_deprived:Subject337   0.7518     5.5703   0.135 0.892895
## days_deprived:Subject349  -5.2644     5.5703  -0.945 0.346731
## days_deprived:Subject350   1.6007     5.5703   0.287 0.774382
## days_deprived:Subject351 -13.1681     5.5703  -2.364 0.019867 *
## days_deprived:Subject352 -14.4019     5.5703  -2.585 0.011057 *
## days_deprived:Subject369  -7.8948     5.5703  -1.417 0.159273
## days_deprived:Subject370  -1.0495     5.5703  -0.188 0.850912
## days_deprived:Subject371  -9.3443     5.5703  -1.678 0.096334 .
## days_deprived:Subject372 -10.6041     5.5703  -1.904 0.059613 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 25.53 on 108 degrees of freedom
## Multiple R-squared:  0.849,Adjusted R-squared:  0.8001
## F-statistic: 17.35 on 35 and 108 DF,  p-value: < 2.2e-16
```

What this model has done is take one subject to be the baseline (specifically, subject 308), and represent each subject in terms of offsets from that baseline. You saw this already when we talked about continuous-by-categorical interactions.

Answer these questions (to three decimal places):

- What is the intercept for subject 308?
- What is the slope for subject 308?

- What is the intercept for subject 335?
- What is the slope for subject 335?

Answers and Explanation

The baseline subject is 308; the default in R is to sort the levels of the factor alphabetically and chooses the first one as the baseline. This means that the intercept and slope for 308 are given by `(Intercept)` and `days_deprived` respectively, because all of the other 17 dummy variables will be zero for subject 308.

All of the regression coefficients for the other subjects are represented as *offsets* from this baseline subject. If we want to calculate the intercept and slope for a given subject, we just add in the corresponding offsets. So, the answers are

- intercept for 308: 288.217
- slope for 308: 21.69
- intercept for 335: `(Intercept)` + `Subject335` = 288.217 + -25.343 = 262.874
- slope for 335: `days_deprived` + `days_deprived:Subject335` = 21.69 + -25.899 = -4.209

In the "no pooling" model, there is no *overall* population intercept and slope that is being estimated; in this case, `(Intercept)` and `days_deprived` are estimates of the intercept and slope for subject 308, which was (arbitrarily) chosen as the baseline subject. To get population estimates, we could introduce a second stage of analysis where we calculate means of the individual intercepts and slopes. Let's use the model estimates to calculate the intercepts and slopes for each subject.

```r
all_intercepts <- c(coef(np_model)["(Intercept)"],
                    coef(np_model)[3:19] + coef(np_model)["(Intercept)"])

all_slopes  <- c(coef(np_model)["days_deprived"],
                 coef(np_model)[20:36] + coef(np_model)["days_deprived"])

ids <- sleep2 %>% pull(Subject) %>% levels() %>% factor()

# make a tibble with the data extracted above
np_coef <- tibble(Subject = ids,
                  intercept = all_intercepts,
                  slope = all_slopes)

np_coef
```

```
## # A tibble: 18 x 3
##    Subject intercept slope
##    <fct>       <dbl> <dbl>
##  1 308          288. 21.7
##  2 309          200.  4.36
##  3 310          226.  3.90
##  4 330          273.  8.01
##  5 331          298.  4.87
##  6 332          316.  2.40
##  7 333          285. 10.9
##  8 334          238. 18.1
##  9 335          263. -4.21
## 10 337          313. 22.4
## 11 349          229. 16.4
## 12 350          248. 23.3
## 13 351          264.  8.52
## 14 352          331.  7.29
## 15 369          267. 13.8
## 16 370          235. 20.6
## 17 371          258. 12.3
## 18 372          291. 11.1
```

Let's see how well this model fits our data.

```
ggplot(sleep2, aes(x = days_deprived, y = Reaction)) +
  geom_abline(data = np_coef,
              mapping = aes(intercept = intercept,
                            slope = slope),
              color = 'blue') +
  geom_point() +
  scale_x_continuous(breaks = 0:7) +
  facet_wrap(~Subject) +
  labs(y = "Reaction Time", x = "Days deprived of sleep (0 = baseline)")
```

This is much better than the complete pooling model. If we want to test the null hypothesis that the fixed slope is zero, we could do using a one-sample test.

```
np_coef %>% pull(slope) %>% t.test()
```

```
##
##  One Sample t-test
##
## data:  .
## t = 6.1971, df = 17, p-value = 9.749e-06
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
##   7.542244 15.328613
## sample estimates:
## mean of x
```

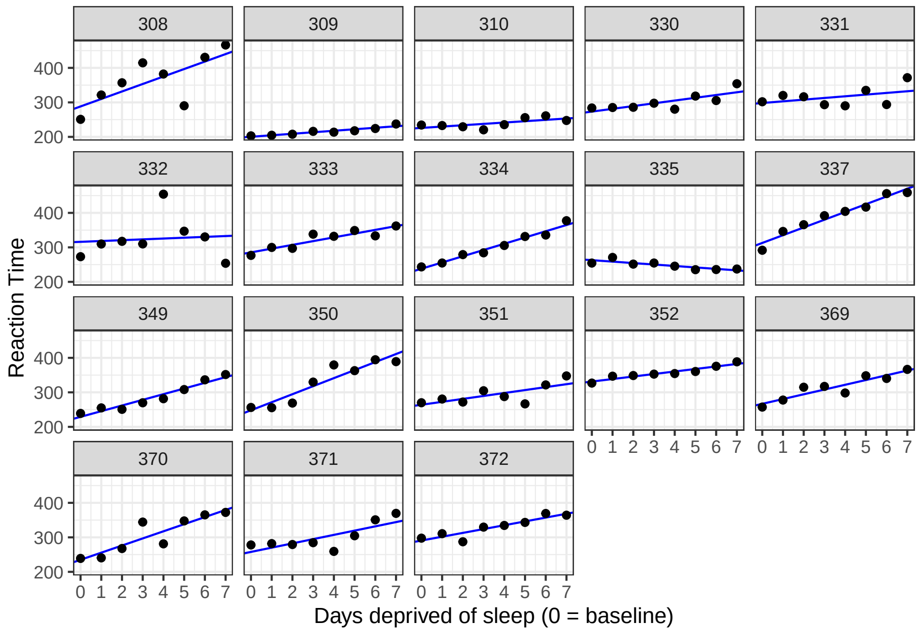

Figure 5.5: Data plotted against fits from the no-pooling approach.

```
##  11.43543
```

This tells us that the mean slope of 11.435 is significantly different from zero, t(17) = 6.197, $p < .001$.

5.3.3 Partial pooling using mixed-effects models

Neither the complete or no-pooling approach is satisfactory. It would be desirable to improve our estimates for individual participants by taking advantage of what we know about the other participants. This will help us better distinguish signal from error for each participant and improve generalization to the population. As the web app below will show, this becomes particularly important when we have unbalanced or missing data.

In the no-pooling model, we treated `Subject` as a **fixed factor**. Each pair of intercept and slope estimates is determined by that subject's data alone. However, we are not interested in these 18 subjects in and of themselves; rather, we are interested in them as examples drawn from a larger population of potential subjects. This subjects-as-fixed-effects approach is suboptimal if your goal is to generalize to new participants in the population of interest.

Partial pooling happens when you treat a factor as a random instead of fixed in your analysis. A **random factor** is a factor whose levels are considered to represent a proper subset of all the levels in the population. Usually, you treat a factor as random when the levels you have in the data are the result of sampling, and you want to generalize beyond those levels. In this case, we have eighteen unique subjects and thus, eighteen levels of the `Subject` factor, and would like to say something general about effects of sleep deprivation on the population of potential subjects.

Just as there are two types of factors—fixed and random—there are

A way to include random factors in your analysis is to use a linear mixed-effects model.

estimates at each level of the factor (i.e., for each subject) become informed by information from other levels (i.e., other subjects). Rather than estimating the intercept and slope for each participant without considering the estimates for other subjects, the model estimates values for the population, and pulls the estimates for individual subjects toward those values, a statistical phenomenon known as `shrinkage`.

The population estimates are informed by the individual estimates, but each individual estimate is "pulled" toward the population value.

The multilevel model is below. It is important that you understand the math and what it means. It looks complicated at first, but there's really nothing below that you haven't seen before. We'll explain everything step by step.

Level 1:

$$Y_{sd} = \beta_{0s} + \beta_{1s} X_{sd} + e_{sd} \tag{5.1}$$

Level 2:

$$\beta_{0s} = \gamma_0 + S_{0s} \tag{5.2}$$

$$\beta_{1s} = \gamma_1 + S_{1s} \tag{5.3}$$

Variance Components:

$$\langle S_{0s}, S_{1s} \rangle \sim N\left(\langle 0, 0 \rangle, \ \right) \tag{5.4}$$

$$= \left(\begin{array}{cc} {\tau_{00}}^2 & \rho\tau_{00}\tau_{11} \\ \rho\tau_{00}\tau_{11} & {\tau_{11}}^2 \end{array} \right) \tag{5.5}$$

$$e_{sd} \sim N\left(0, \sigma^2\right) \tag{5.6}$$

In case you get lost, here's a table with an explanation for all of the variables in the set of equations above.

Variable	Type	Description
Y_{sd}	observed	Value of `Reaction` for subject s on day d
X_{sd}	observed	Value of `Day` (0-9) for subject s on day d
β_{0s}	derived	level 1 intercept parameter
β_{1s}	derived	level 1 slope parameter
e_{sd}	derived	Residual (Y_{sd} - $\hat{Y}_{sd}$) for subject s, day d
γ_0	fixed	Grand intercept ("gamma")
γ_1	fixed	Grand slope ("gamma")
S_{0s}	derived	Random intercept (offset) for subject s
S_{1s}	derived	Random slope (offset) for subject s
	random	Variance-covariance matrix
${\tau_{00}}^2$	random	Variance of random intercepts
ρ	random	Random correlation between intercepts and slopes
${\tau_{11}}^2$	random	Variance of random slopes
σ^2	random	Error variance

Note the "Status" column of the table contains values *fixed*, *random*, and *derived*. Although *fixed* and *random* are standard terms, *derived* is not; I have introduced it to help you think about what these different variables mean in the context of the model and to help you distinguish variables that are directly estimated from variables that are not.

Let's begin with the Level 1 equation of our model, which represents the general relationship between the predictors and response variable. It captures the functional form of the main relationship between reaction time Y_{sd} and sleep deprivation X_{sd}: a straight line with intercept β_{0s} and slope β_{1s}. Now β_{0s} and β_{1s} makes it look like the complete pooling model, where we estimated a single intercept and single slope for the entire dataset; however, we're not actually estimating these directly. Instead, we're going to think of β_{0s} and β_{1s} as derived: they are wholly defined by variables at Level 2 of the model.

Level 2 of the model, defined by two equations, represents relationships at the participant level. Here, we define the intercept β_{0s} in terms of a fixed effect γ_0 and a *random intercept* S_{0s}; likewise, we define the slope β_{1s} in terms of a fixed slope γ_1 and a *random slope* S_{1s}.

The final equations represent the *Variance Components* of the model. We'll get into this more in detail below.

Let's substitute the Level 2 equations into Level 1 to see the advantages of representing things in the multilevel way.

$$Y_{sd} = \gamma_0 + S_{0s} + (\gamma_1 + S_{1s})\, X_{sd} + e_{sd} \tag{5.7}$$

While this "combined" formula syntax is easy enough to understand in this particular case, the multilevel form more clearly allows us to see the functional form of the model: a straight line. We could easily change the functional form to, for instance, capture non-linear trends:

$$Y_{sd} = \beta_{0s} + \beta_{1s} X_{sd} + \beta_{2s} X_{sd}^2 + e_{sd}$$

This functional form gets obscured in the combined syntax. The multilevel syntax also makes it easy to see which terms go with the intercept and which terms go with the slope. Also, as designs get more compilicated—for example, if we were to assign participants to different experimental conditions, thus introducing a further predictors at Level 2—the combined equations get harder and harder to parse and reason about.

Fixed effect parameters like γ_0 and γ_1 are estimated from the data, and reflect stable properties of the population. In this example, γ_0 is the population intercept and γ_1 is the population slope. You can think of these **fixed-effects parameters** as representing the **average intercept and slope** in the population. These are "fixed" in the sense that we assume that they reflect the true underlying values in the population; they are not assumed to vary from sample to sample. These fixed effects parameters are often of prime theoretical interest; we want to measure them and their standard errors in a manner that is as unbiased and precise as the data allow. In experimental settings they are often the targets of hypothesis tests.

Random effects like S_{0i} and S_{1i} allow intercepts and slopes (respectively) to vary over subjects. These random effects are *offsets*: deviations from the population 'grand mean' values. Some subjects will just be slower responders than others, such that they will have a higher intercept (mean RT) on day 0 than the population's estimated value of $\hat{\gamma_0}$. These slower-than-average subjects will have positive S_{0i} values; faster-than-average subjects will have negative S_{0i} values. Likewise, some subjects will show stronger effects of sleep deprivation (steeper slope) than the estimated population effect, $\hat{\gamma_1}$, which implies a positive offset S_{1s}, while others may show weaker effects or close to none at all (negative offset).

Each participant can be represented as a vector pair $\langle S_{0i}, S_{1i} \rangle$. If the subjects in our sample comprised the entire population, we would be justified in treating them as fixed and estimating their values, as in the "no-pooling" approach above. This is not the case here. In recognition of the fact they are sampled, we are going to treat subjects as a random factor rather than a fixed factor. Instead of estimate the values for the subjects we happened to pick, we will estimate **the covariance matrix that represents the bivariate distribution from which these pairs of values are drawn**. By doing this, we allow the subjects in the sample to inform us about characteristics of the population.

5.3.4 The variance-covariance matrix

$$\langle S_{0s}, S_{1s} \rangle \sim N\left(\langle 0, 0 \rangle, \ \right) \tag{5.8}$$

$$= \begin{pmatrix} {\tau_{00}}^2 & \rho\tau_{00}\tau_{11} \\ \rho\tau_{00}\tau_{11} & {\tau_{11}}^2 \end{pmatrix} \tag{5.9}$$

Equations in the *Variance Components* characterize our estimates of variability. The first equation states our assumption that the random intercept / random slope pairs $\langle S_{0s}, S_{1s} \rangle$ are drawn from a bivariate normal distribution centered at the origin $\langle 0, 0 \rangle$ with variance-covariance matrix .

The variance-covariance matrix is key: it determines the probability of drawing random effect pairs $\langle S_{0s}, S_{1s} \rangle$ from the population. You have seen these before, in the chapter on correlation and regression. The covariance matrix is always a square matrix (equal numbers of columns and rows). On the main diagonal (upper left and bottom right cells) it has random effect variances ${\tau_{00}}^2$ and ${\tau_{11}}^2$. ${\tau_{00}}^2$ is the random intercept variance, which captures how much subjects vary in their mean response time on Day 0, before any sleep deprivation. ${\tau_{11}}^2$ is the random slope variance, which captures how much subjects vary in their susceptibility to the effects of sleep deprivation.

The cells in the off-diagonal contain covariances, but this information is represented redundantly in the matrix; the lower left element is identical to the upper right element; both capture the covariance between random intercepts and slopes, as expressed by $\rho\tau_{00}\tau_{11}$. In this equation ρ is the correlation between the intercept and slope. So, all the information in the matrix can be captured by just three parameters: τ_{00}, τ_{11}, and ρ.

5.4 Estimating the model parameters

To estimate parameters, we are going to use the `lmer()` function of the lme4 package (Bates, Mächler, Bolker, & Walker, 2015). The basic syntax of `lmer()` is

```
lmer(formula, data, ...)
```

where `formula` expresses the structure of the underlying model in a compact format and `data` is the data frame where the variables mentioned in the formula can be found.

The general format of the model formula for N fixed effects (`fix`) and K random effects (`ran`) is

```
DV ~ fix1 + fix2 + ... + fixN + (ran1 + ran2 + ... + ranK |
random_factor1)
```

Interactions between factors A and B can be specified using either `A * B` (interaction and main effects) or `A:B` (just the interaction).

A key difference from standard R model syntax is the presence of a random effect term, which is enclosed in parentheses, e.g., `(ran1 + ran2 + ... + ranK`

`| random_factor)`. Each bracketed expression represents random effects associated with a single random factor. You can have more than one random effects term in a single formula, as we will see when we talk about crossed random factors. You should think of the random effects terms as providing `lmer()` with **instructions on how to construct variance-covariance matrices**.

On the left side of the bar `|` you put the effects you want to allow to vary over the levels of the random factor named on the right side. Usually, the right-side variable is one whose values uniquely identify individual subjects (e.g., `subject_id`).

Consider the following possible model formulas for the `sleep2` data and the variance-covariance matrices they construct.

	model	syntax
1	random intercepts only	`Reaction ~ days_deprived + (1 | Subject)`
2	random intercepts and slopes	`Reaction ~ days_deprived + (1 + days_deprived | Subject)`
3	model 2 alternate syntax	`Reaction ~ days_deprived + (days_deprived | Subject)`
4	random slopes only	`Reaction ~ days_deprived + (0 + days_deprived | Subject)`
5	model 2 + zero-covariances	`Reaction ~ days_deprived + (days_deprived || Subject)`

Model 1:

$$= \begin{pmatrix} {\tau_{00}}^2 & 0 \\ 0 & 0 \end{pmatrix}$$

Models 2 and 3:

$$= \begin{pmatrix} {\tau_{00}}^2 & \rho\tau_{00}\tau_{11} \\ \rho\tau_{00}\tau_{11} & {\tau_{11}}^2 \end{pmatrix}$$

Model 4:

$$= \begin{pmatrix} 0 & 0 \\ 0 & {\tau_{11}}^2 \end{pmatrix}$$

Model 5:

$$= \begin{pmatrix} {\tau_{00}}^2 & 0 \\ 0 & {\tau_{11}}^2 \end{pmatrix}$$

The most reasonable model for these data is Model 2, so we'll stick with that.

Let's fit the model, storing the result in object `pp_mod`.

```
pp_mod <- lmer(Reaction ~ days_deprived + (days_deprived | Subject), sleep2)

summary(pp_mod)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: Reaction ~ days_deprived + (days_deprived | Subject)
##    Data: sleep2
##
## REML criterion at convergence: 1404.1
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.0157 -0.3541  0.0069  0.4681  5.0732
##
## Random effects:
##  Groups   Name          Variance Std.Dev. Corr
##  Subject  (Intercept)   958.35   30.957
##           days_deprived  45.78    6.766   0.18
##  Residual               651.60   25.526
## Number of obs: 144, groups:  Subject, 18
##
## Fixed effects:
##               Estimate Std. Error t value
## (Intercept)    267.967      8.266  32.418
## days_deprived   11.435      1.845   6.197
##
## Correlation of Fixed Effects:
##             (Intr)
## days_deprvd -0.062
```

Before discussing how to interpret the output, let's first plot the data against our model predictions. We can get model predictions using the `predict()` function (see `?predict.merMod` for information about use with mixed-effects models).

First, create a new data frame with predictor values for `Subject` and `days_deprived`.

```
newdata <- crossing(
  Subject = sleep2 %>% pull(Subject) %>% levels() %>% factor(),
  days_deprived = 0:7)

newdata
```

```
## # A tibble: 144 x 2
##    Subject days_deprived
##    <fct>           <int>
##  1 308                 0
##  2 308                 1
##  3 308                 2
##  4 308                 3
##  5 308                 4
##  6 308                 5
##  7 308                 6
##  8 308                 7
##  9 309                 0
## 10 309                 1
## # ... with 134 more rows
```

Then, run this through `predict()`. Typically we will add the prediction in as a new variable in the data frame of new data, giving it the same name as our DV (`Reaction`).

```
newdata2 <- newdata %>%
  mutate(Reaction = predict(pp_mod, newdata))
```

Now we are ready to plot.

```
ggplot(sleep2, aes(x = days_deprived, y = Reaction)) +
  geom_line(data = newdata2,
            color = 'blue') +
  geom_point() +
  scale_x_continuous(breaks = 0:7) +
  facet_wrap(~Subject) +
  labs(y = "Reaction Time", x = "Days deprived of sleep (0 = baseline)")
```

5.5 Interpreting `lmer()` output and extracting estimates

The call to `lmer()` returns a fitted model object of class "lmerMod". To find out more about the `lmerMod` class, which is in turn a specialized version of the `merMod` class, see `?lmerMod-class`.

5.5.1 Fixed effects

The section of the output called `Fixed effects:` should look familiar; it is similar to what you would see in the output for a simple linear model fit by `lm()`.

```
## Fixed effects:
##               Estimate Std. Error t value
## (Intercept)    267.967      8.266  32.418
## days_deprived   11.435      1.845   6.197
```

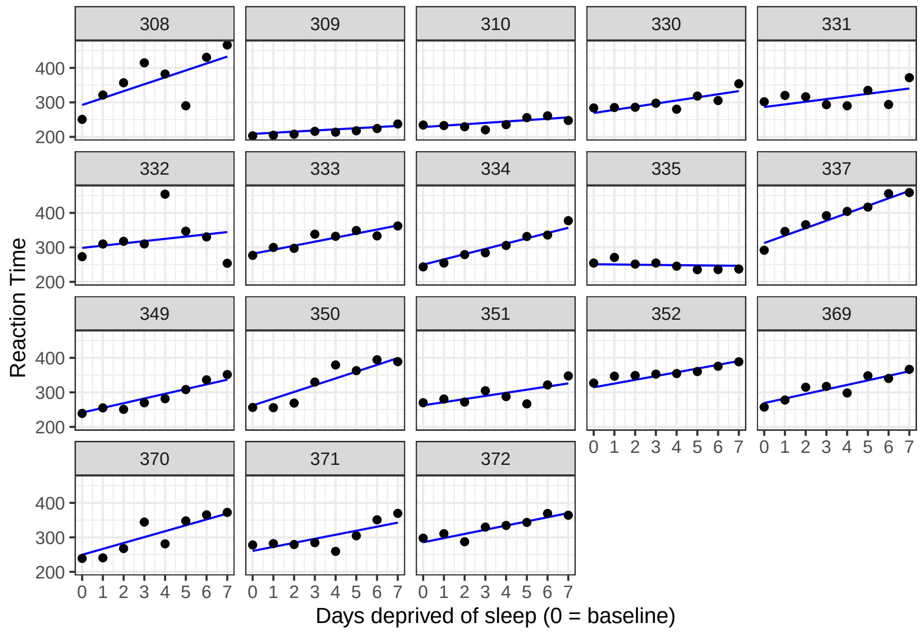

Figure 5.6: Data plotted against predictions from a partial pooling approach.

This indicates that the estimated mean reaction time for participants at Day 0 was about 268 milliseconds, with each day of sleep deprivation adding an additional 11 milliseconds to the response time, on average.

If we need to get the fixed effects from the model, we can extract them using `fixef()`.

```
fixef(pp_mod)
```

```
##   (Intercept) days_deprived
##     267.96742      11.43543
```

The standard errors give us estimates of the variability for these parameters due to sampling error. You could use these to calculate the t-values or derive confidence intervals. Extract them using `vcov(pp_mod)` which gives a variance-covariance matrix (*not* the one associated with the random effects), pull out the diagonal using `diag()` and then take the square root using `sqrt()`.

```
sqrt(diag(vcov(pp_mod)))

# OR, equivalently using pipes:
# vcov(pp_mod) %>% diag() %>% sqrt()
```

```
## [1] 8.265896 1.845293
```

Note that these t values do not appear with p values, as is customary in simpler modeling frameworks. There are multiple approaches for getting p values from mixed-effects models, with advantages and disadvantages to each; see Luke (2017) for a survey of options. The t values do not appear with degrees of freedom, because the degrees of freedom in a mixed-effects model are not well-defined. Often people will treat them as Wald z values, i.e., as observations from the standard normal distribution. Because the t distribution asymptotes the standard normal distribution as the number of observations goes to infinity, this "t-as-z" practice is legitimate if you have a large enough set of observations.

To calculate the Wald z values, just divide the fixed effect estimate by its standard error:

```
tvals <- fixef(pp_mod) / sqrt(diag(vcov(pp_mod)))

tvals
```

```
##   (Intercept) days_deprived
##     32.418437      6.197082
```

You can get the associated p-values using the following formula:

```
2 * (1 - pnorm(abs(tvals)))
```

```
##   (Intercept) days_deprived
##   0.00000e+00   5.75197e-10
```

This gives us strong evidence against the null hypothesis $H_0 : \gamma_1 = 0$. Sleep deprivation does appear to increase response time.

You can get confidence intervals for the estimates using `confint()` (this technique uses the *parametric bootstrap*). `confint()` is a generic function, so to get help on this function, use `?confint.merMod`.

```
confint(pp_mod)
```

```
## Computing profile confidence intervals ...

##                      2.5 %      97.5 %
## .sig01          19.0979934  46.3366599
## .sig02          -0.4051073   0.8058951
## .sig03           4.0079284  10.2487351
## .sigma          22.4666029  29.3494509
## (Intercept)    251.3443396 284.5904989
## days_deprived    7.7245247  15.1463328
```

5.5.2 Random effects

```
## Random effects:
##  Groups   Name          Variance Std.Dev. Corr
##  Subject  (Intercept)   958.35   30.957
##            days_deprived  45.78    6.766   0.18
##  Residual                651.60   25.526
## Number of obs: 144, groups:  Subject, 18
```

The random effects part of the `summary()` output is less familiar. What you find here is a table with information about the variance components: the variance-covariance matrix (or matrices, if you have multiple random factors) and the residual variance.

Let's start with the `Residual` line. This tells us that the residual variance, σ^2, was estimated at about 651.6. The value in the next column, 25.526, is just the standard deviation, σ, which is the square root of the variance.

We extract the residual standard deviation using the `sigma()` function.

```
sigma(pp_mod) # residual
```

```
## [1] 25.5264
```

The two lines above the `Residual` line give us information about the variance-covariance matrix for the `Subject` random factor.

```
##  Groups   Name          Variance Std.Dev. Corr
##  Subject  (Intercept)   958.35   30.957
##           days_deprived  45.78    6.766   0.18
```

The values in the `Variance` column gives us the main diagonal of the matrix, and the `Std.Dev.` values are just the square roots of these values. The `Corr` column tells us the correlation between the intercept and slope.

We can extract these values from the fitted object `pp_mod` using the `VarCorr()` function. This returns a named list, with one element for each random factor. We have `Subject` as our only random factor, so the list will just be of length 1.

```
# variance-covariance matrix for random factor Subject
VarCorr(pp_mod)[["Subject"]] # equivalently: VarCorr(pp_mod)[[1]]
```

```
##               (Intercept) days_deprived
## (Intercept)      958.3517      37.20460
## days_deprived     37.2046      45.77766
## attr(,"stddev")
##   (Intercept) days_deprived
##     30.957255      6.765919
## attr(,"correlation")
##               (Intercept) days_deprived
## (Intercept)     1.0000000     0.1776263
## days_deprived   0.1776263     1.0000000
```

The first few lines are a printout of the variance covariance matrix. You can see the variances in the main diagonal. We can get these with:

```
diag(VarCorr(pp_mod)[["Subject"]]) # just the variances
```

```
##   (Intercept) days_deprived
##     958.35165      45.77766
```

We can get the correlation between the intecepts and slopes in two ways. First, by extracting the `"correlation"` attribute and then pulling out the element in row 1 column 2 (`[1, 2]`):

```
attr(VarCorr(pp_mod)[["Subject"]], "correlation")[1, 2] # the correlation
```

```
## [1] 0.1776263
```

Or we can directly compute the value from the variance-covariance matrix itself.

```
# directly compute correlation from variance-covariance matrix
mx <- VarCorr(pp_mod)[["Subject"]]

## if cov = rho * t00 * t11, then
## rho = cov / (t00 * t11).
mx[1, 2] / (sqrt(mx[1, 1]) * sqrt(mx[2, 2]))
```

```
## [1] 0.1776263
```

We can pull out the estimated random effects (BLUPS) using `ranef()`. Like `VarCorr()` , the result is a named list, with each element corresponding to a single random factor.

```
ranef(pp_mod)[["Subject"]]
```

```
##     (Intercept) days_deprived
## 308  24.4992891     8.6020000
## 309 -59.3723102    -8.1277534
## 310 -39.4762764    -7.4292365
## 330   1.3500428    -2.3845976
## 331  18.4576169    -3.7477340
## 332  30.5270040    -4.8936899
## 333  13.3682027     0.2888639
## 334 -18.1583020     3.8436686
## 335 -16.9737887   -12.0702333
## 337  44.5850842    10.1760837
## 349 -26.6839022     2.1946699
## 350  -5.9657957     8.1758613
## 351  -5.5710355    -2.3718494
## 352  46.6347253    -0.5616377
## 369   0.9616395     1.7385130
## 370 -18.5216778     5.6317534
## 371  -7.3431320     0.2729282
## 372  17.6826159     0.6623897
```

There are other extractor functions that are useful. See `?merMod-class` for details.

We can get fitted values from the model using `fitted()` and residuals using `residuals()`. (These functions take into account "the conditional modes of the random effects", i.e., the BLUPS).

```
mutate(sleep2,
       fit = fitted(pp_mod),
       resid = residuals(pp_mod)) %>%
  group_by(Subject) %>%
  slice(c(1,10)) %>%
  print(n = +Inf)
```

```
## # A tibble: 18 x 6
## # Groups:   Subject [18]
##    Reaction  Days Subject days_deprived   fit  resid
##       <dbl> <dbl> <fct>           <dbl> <dbl>  <dbl>
##  1     251.     2 308                 0  292. -41.7
##  2     203.     2 309                 0  209.  -5.62
##  3     234.     2 310                 0  228.   5.83
##  4     284.     2 330                 0  269.  14.5
##  5     302.     2 331                 0  286.  15.4
##  6     273.     2 332                 0  298. -25.5
##  7     277.     2 333                 0  281.  -4.57
##  8     243.     2 334                 0  250.  -6.44
##  9     254.     2 335                 0  251.   3.50
## 10     292.     2 337                 0  313. -20.9
## 11     239.     2 349                 0  241.  -2.36
## 12     256.     2 350                 0  262.  -5.80
## 13     270.     2 351                 0  262.   7.50
## 14     327.     2 352                 0  315.  12.3
## 15     257.     2 369                 0  269. -11.7
## 16     239.     2 370                 0  249. -10.5
## 17     278.     2 371                 0  261.  17.3
## 18     298.     2 372                 0  286.  11.9
```

Finally, we can get predictions for new data using `predict()`, as we did above. Below we use `predict()` to imagine what might have happened had we continued our study for three extra days.

```
## create the table with new predictor values
ndat <- crossing(Subject = sleep2 %>% pull(Subject) %>% levels() %>% factor(),
                 days_deprived = 8:10) %>%
  mutate(Reaction = predict(pp_mod, newdata = .))
```

```
ggplot(sleep2, aes(x = days_deprived, y = Reaction)) +
  geom_line(data = bind_rows(newdata2, ndat),
            color = 'blue') +
  geom_point() +
  scale_x_continuous(breaks = 0:10) +
  facet_wrap(~Subject) +
  labs(y = "Reaction Time", x = "Days deprived of sleep (0 = baseline)")
```

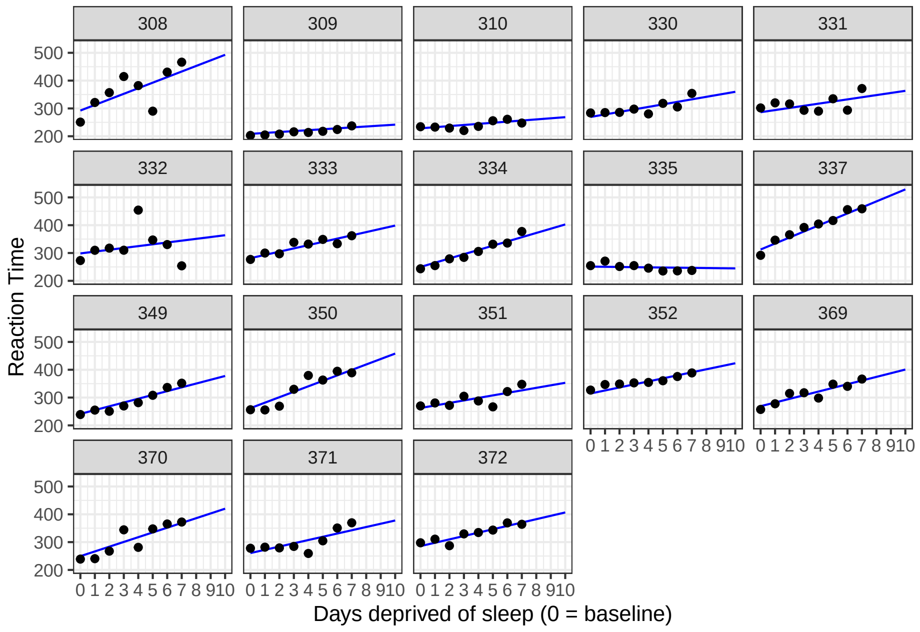

Figure 5.7: Data against model with extrapolation.

5.6 Multi-level app

Try out the multi-level web app to sharpen your understanding of the three different approaches to multi-level modeling.

Chapter 6

Linear mixed-effects models with one random factor

6.1 Learning objectives

- understand how linear mixed-effects models can replace conventional analyses, and when they are appropriate
- express various types of common designs in a regression framework
- use model comparison (`anova()`) for testing effects
- express various study designs using the R regression formula syntax

6.2 When, and why, would you want to replace conventional analyses with linear mixed-effects modeling?

We have repeatedly emphasized how many common techniques in psychology can be seen as special cases of the general linear model. This implies that it would be possible to replace these techniques with regression. In fact, you could analyze almost any conceivable dataset in psychology using one of the four functions below:

sampling design	type of data	function	description
single level	continuous, normally distributed	‘base::lm()‘	simple linear model
single level	count or categorical	‘base::glm()‘	generalized linear model
multilevel	continuous, normally distributed	‘lme4::lmer()‘	linear mixed-effects model
multilevel	count or categorical	‘lme4::glmer()‘	generalized linear mixed-effects model

To decide which function to use, you need to know the type of data you're working with (continuous and normally distributed, or not) and how the data have been sampled (single-level or multilevel). Arguments to these functions are highly similar across all four versions. We will learn about analyzing count and categorical data later in this course. For now, we will focus on continuous data, but many of the principles are identical.

Here is a comparison chart for single-level data (data where you don't have repeated-measures):

test	conventional approach	regression approach
one-sample t-test	't.test(y, mu = c)'	'lm(y ~ 1, offset = c)'
independent samples t-test	't.test(x, y)'	'lm(y ~ x)'
one factor ANOVA	'aov(y ~ x)'	'lm(y ~ x)'
factorial ANOVA	'aov(y ~ a * b)'	'lm(y ~ a * b)'

All of above designs are *between-subjects* designs without repeated measures. (Note that in the factorial case, we would probably replace `a` and `b` with our own deviation-coded numerical predictors, for reasons already discussed in the chapter on interactions).

Where mixed-effects models come into play is with multilevel data. Data is usually multilevel for one of the three reasons below (multiple reasons could simultaneously apply):

1. you have a within-subject factor, and/or
2. you have **pseudoreplications**, and/or
3. you have multiple stimulus items (which we will discuss in the next chapter).

(At this point, it would be a good idea to refresh your memory on the meaning of between- versus within- subject factors). In the `sleepstudy` data, you had the within-subject factor of `Day` (which is more a numeric variable, actually, than a factor; but it has multiple values varying within each participant).

Pseudoreplications occur when you take multiple measurements within the same condition. For instance, imagine a study where you randomly assign participants to consume one of two beverages—alcohol or water—before administering a simple response time task where they press a button as fast as possible when a light flashes. You would probably take more than one measurement of response time for each participant; let's assume that you measured it over 100 trials. You'd have one between-subject factor (beverage) and 100 observations per subject, for say, 20 subjects in each group. One common mistake novices make when analyzing such data is to try to run a t-test. **You can't directly use the conventional a t-test when you have pseudoreplications (or multiple stimuli).** You must first calculate means for each subject, and then run your analysis **on the means, not on the raw data.** There are versions of ANOVA that can deal with pseudoreplications, but you are probably better off using a linear-mixed effects model, which can better handle the complex dependency structure.

Here is a comparison chart for multi-level data:

test	conventional approach	regr
one-sample t-test with pseudoreplications	calculate means and use ‘t.test(x_mean)‘	<co
paired samples t-test, no pseudoreplications	‘t.test(x, y, paired = TRUE)‘	<co
paired samples t-test with pseudoreplications	calculate means and use ‘t.test(x_mean, y_mean)‘	<co
repeated-measures ANOVA no pseudoreplications	‘aov(y ~ x + Error(subject))‘	<co
repeated-measures ANOVA with pseudoreplications	‘aov(y ~ x + Error(subject/x))‘	<co
factorial ANOVA, a & b within, no pseudoreplications	‘aov(y ~ a * b + Error(subject))‘	<co
factorial ANOVA with pseudoreplications	‘aov(y ~ a * b + Error(subject/(a * b))‘	<co

One of the main selling points of the general linear models / regression framework over t-test and ANOVA is its flexibility. We saw this in the last chapter with the `sleepstudy` data, which could only be properly handled within a linear mixed-effects modelling framework. Despite the many advantages of regression, if you are in a situation where you have balanced data and can reasonably apply t-test or ANOVA without violating any of the assumptions behind the test, it makes sense to do so; these approaches have a long history in psychology and are more widely understood.

6.3 Example: Independent-samples *t*-test on multi-level data

Let’s consider a situation where you are testing the effect of alcohol consumption on simple reaction time (e.g., press a button as fast as you can after a light appears). To keep it simple, let’s assume that you have collected data from 14 participants randomly assigned to perform a set of 10 simple RT trials after one of two interventions: drinking a pint of alcohol (treatment condition) or a placebo drink (placebo condition). You have 7 participants in each of the two groups. Note that you would need more than this for a real study.

This web app presents simulated data from such a study. Subjects P01-P07 are from the placebo condition, while subjects T01-T07 are from the treatment condition. Please stop and have a look!

If we were going to run a t-test on these data, we would first need to calculate subject means, because otherwise the observations are not independent. You could do this as follows. (If you want to run the code below, you can download sample data from the web app above and save it as `independent_samples.csv`).

```r
library("tidyverse")

dat <- read_csv("data/independent_samples.csv", col_types = "cci")

subj_means <- dat %>%
  group_by(subject, cond) %>%
  summarise(mean_rt = mean(RT)) %>%
  ungroup()

subj_means
```

```
## # A tibble: 14 x 3
##    subject cond  mean_rt
##    <chr>   <chr>   <dbl>
##  1 P01     P        354
##  2 P02     P        384.
##  3 P03     P        391.
##  4 P04     P        404.
##  5 P05     P        421.
##  6 P06     P        392
##  7 P07     P        400.
##  8 T08     T        430.
##  9 T09     T        432.
## 10 T10     T        410.
## 11 T11     T        455.
## 12 T12     T        450.
## 13 T13     T        418.
## 14 T14     T        489.
```

Then, the t-test can be run using the "formula" version of `t.test()`.

```
t.test(mean_rt ~ cond, subj_means)
```

```
##
##  Welch Two Sample t-test
##
## data:  mean_rt by cond
## t = -3.7985, df = 11.32, p-value = 0.002807
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -76.32580 -20.44563
## sample estimates:
## mean in group P mean in group T
##        392.3143        440.7000
```

While there is nothing wrong with this analysis, aggregating the data throws away information. we can see in the above web app that there are actually two different sources of variability: trial-by-trial variability in simple RT (represented by σ) and variability across subjects in terms of their how slow or fast they are relative to the population mean (γ_{00}). The Data Generating Process for response time (Y_{st}) for subject s on trial t is shown below.

Level 1:

$$Y_{st} = \beta_{0s} + \beta_1 X_s + e_{st} \tag{6.1}$$

Level 2:

$$\beta_{0s} = \gamma_{00} + S_{0s} \tag{6.2}$$

$$\beta_1 = \gamma_{10} \tag{6.3}$$

Variance Components:

$$S_{0s} \sim N\left(0, {\tau_{00}}^2\right) \tag{6.4}$$

$$e_{st} \sim N\left(0, \sigma^2\right) \tag{6.5}$$

In the above equation, X_s is a numerical predictor coding which condition the subject s is in; e.g., 0 for placebo, 1 for treatment.

The multi-level equations are somewhat cumbersome for such a simple model; we could just reduce levels 1 and 2 to

$$Y_{st} = \gamma_{00} + S_{0s} + \gamma_{10} X_s + e_{st}, \tag{6.6}$$

but it is worth becoming familiar with the multi-level format for when we encounter more complex designs.

Unlike the `sleepstudy` data seen in the last chapter, we only have one random effect for each subject, S_{0s}. There is no random slope. Each subject appears in only one of the two treatment conditions, so it would not be possible to estimate how the effect of placebo versus alcohol varies over subjects. The mixed-effects model that we would fit to these data, with random intercepts but no random slopes, is known as a **random intercepts model**.

A random-intercepts model would adequately capture the two sources of variability mentioned above: the inter-subject variability in overall mean RT in the parameter ${\tau_{00}}^2$, and the trial-by-trial variability in the parameter σ^2. We can calculate the proportion of the total variability attributable to individual differences among subjects using the formula below.

$$ICC = \frac{{\tau_{00}}^2}{{\tau_{00}}^2 + \sigma^2}$$

This quantity, known as the **intra-class correlation coefficient**, and tells you how much clustering there is in your data. It ranges from 0 to 1, with 0 indicating that all the variability is due to residual variance, and 1 indicating that all the variability is due to individual differences among subjects.

The lmer syntax for fitting a random intercepts model to the data is `lmer(RT ~ cond + (1 | subject), dat, REML=FALSE)`. Let's create our own numerical predictor first, to make it explicit that we are using dummy coding.

```
dat2 <- dat %>%
  mutate(cond_d = if_else(cond == "T", 1L, 0L))

distinct(dat2, cond, cond_d)  ## double check
```

```
## # A tibble: 2 x 2
##   cond  cond_d
##   <chr>  <int>
## 1 P          0
## 2 T          1
```

And now, estimate the model.

```
library("lme4")

mod <- lmer(RT ~ cond_d + (1 | subject), dat2, REML = FALSE)

summary(mod)
```

```
## Linear mixed model fit by maximum likelihood  ['lmerMod']
## Formula: RT ~ cond_d + (1 | subject)
##    Data: dat2
##
##      AIC      BIC   logLik deviance df.resid
##   1451.8   1463.5   -721.9   1443.8      136
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.67117 -0.66677  0.01656  0.75361  2.58447
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subject  (Intercept)  329.3   18.15
##  Residual             1574.7   39.68
## Number of obs: 140, groups:  subject, 14
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  392.314      8.339  47.045
## cond_d        48.386     11.793   4.103
##
```

```
## Correlation of Fixed Effects:
##        (Intr)
## cond_d -0.707
```

Play around with the sliders in the app above and check the lmer output panel until you understand how the output maps onto the model parameters.

6.3.1 When is a random-intercepts model appropriate?

Of course, a mixed-effects model is only appropriate when you have multilevel data. The random-intercepts model is appropriate for any one-sample or between-subjects data with pseudoreplications (if you don't have pseudoreplications in this situation, you don't have multilevel data, and you can just use vanilla regression, e.g., `lm()`).

The "random-intercepts-only" model is also appropriate when you have within-subject factors in your design, but **only if you don't also have pseudoreplications**; that is, it is **only** appropriate when you have a single observation per subject per level of the within-subject factor. If you have more than one observation per subject per level/cell, you need to enrich your random effects structure with random slopes, as described in the next section. If the reason you have multiple observations per subject per level is because you have each subject reacting to the same set of stimuli, then you might want to consider a mixed-effects model with crossed random effects for subjects and stimuli, as described in the next chapter.

The same logic goes for factorial designs in which there is more than one within-subjects factor. In factorial designs, the random-intercepts model is appropriate if you have one observation per subject per **cell** formed by each combination of the within-subjects factors. For instance, if A and B are two two-level within-subject factors, you need to check that you have only one observation for each subject in A_1B_1, A_1B_2, A_2B_1, and A_2B_2. If you have more than one observation, you will need to consider including a random slope in your model.

6.4 Expressing the study design and performing tests in regression

In order to reproduce all of the t-test/ANOVA-style analyses in linear mixed-effects models, you'll need to better understand two things: (1) how to express your study design in a regression formula, and (2) how to get p-values for any tests you perform. The latter part is not obvious within the linear mixed-effects approach, since in many circumstances, the output of `lme4::lmer()` does not give you p-values by default, reflecting the fact that there are multiple options for deriving them (Luke, 2017). Or, you might get p-values for individual regression coefficients, but the test you want to perform is a composite one where you need to test multiple parameters simultaneously.

First, let's take a closer look at the regression formula syntax in R and `lme4`. For the regression model $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_m x_m + e$ with by-subject random effects you'd have:

```
y ~ 1 + x1 + x2 + ... + xm + (1 + x1 + ... | subject)
```

The residual term is implied and so not mentioned; only the predictors. The left side (before the tilde `~`) specifies the response variable, the right side has the predictor variables. The term in brackets, `(... | ...)`, is specific to `lme4`. The right side of the vertical bar `|` specifies the name of a variable (in this case, `subject`) that codes the levels of the random factor. The left side specifies what regression coefficients you want to allow to vary over those levels. The `1` at the start of the formula specifies that you want an intercept, which is included by default anyway, and so can be omitted. The `1` within the bracket specifies a **random intercept**, which is also included by default; the predictor variables mentioned inside the brackets specify **random slopes**. So you could write the above formula equivalently as:

```
y ~ x1 + x2 + ... + xm + (x1 + ... | subject).
```

There is another important type of shortcut in R formulas, which we already saw in the chapter on interactions; namely the "star syntax" `a * b` for specifying interactions. If you have two factors, `a` and `b` in your design, and you want all main effects and interactions in the model, you can use:

```
y ~ a * b
```

which is equivalent to `y ~ a + b + a:b` where `a` and `b` are the predictors coding main effects of A and B respectively, and `a:b` codes the AB interaction. It saves a lot of typing (and mistakes) to use the star syntax for interactions rather than spelling them out. This can be seen more easily in the example below, which codes a 2x2x2 factorial design including factors A, B, and C:

```
y ~ a * b * c
```

which is equivalent to

```
y ~ a + b + c + a:b + a:c + b:c + a:b:c
```

where `a:b`, `a:c`, and `b:c` are two-way interactions and `a:b:c` is the three-way interaction. You can use the star syntax inside the brackets for the random effects term as well, e.g.,

```
y ~ a * b * c + (a * b * c | subject)
```

is equivalent to

```
y ~ a + b + c + a:b + a:c + b:c + a:b:c + (a + b + c + a:b + a:c
+ b:c + a:b:c | subject)
```

which, despite its complexity, is a design that is not that uncommon in psychology (e.g., if you have all factors within and multiple stimuli per condition)!

6.4.1 Factors with more than two levels

As noted above, you could conduct a one-factor ANOVA in regression using the formula `lm(y ~ x)` for single-level or `lmer(y ~ x + (1 | subject))` for the multilevel version without pseudoreplications. In the formula, the `x` predictor would be of type `factor()`, which R would (by default) convert to $k-1$ dummy-coded numerical predictors (one for each level; see the interaction chapter for information).

It is often sensible to code your own numerical predictors, particularly if your goal is to do ANOVA-style tests of main effects and interactions, which can be difficult if any of your variables are of type `factor`. So for a design with one three-level factor called `meal` (`breakfast`, `lunch`, `dinner`) you could create two variables, `lunch_v_breakfast` and `dinner_v_breakfast` following the scheme below.

factor level	lunch_v_breakfast	dinner_v_breakfast
breakfast	-1/3	-1/3
lunch	+2/3	-1/3
dinner	-1/3	+2/3

If your dependent variable is `calories`, your model would be:

`calories ~ lunch_v_breakfast + dinner_v_breakfast`.

But what if you wanted to interact `meal` with another two-level factor—say, `time_of_week` (`weekday` versus `weekend`, coded as -.5 and +.5 respectively), because you think the calories consumed per meal would differ across the levels of this variable? Then your model would be:

`calories ~ (lunch_v_breakfast + dinner_v_breakfast) * time_of_week`.

(The inclusion of an interaction is the reason we chose the "deviation" coding scheme.). We put brackets around the predictors associated with the two-level variable so that each one interacts with `time_of_week`. The above star syntax is shorthand for:

```
calories ~ lunch_v_breakfast + dinner_v_breakfast + time_of_week
+ lunch_v_breakfast:time_of_week + dinner_v_breakfast:time_of_week.
```

This is the "regression way" of estimating parameters for a 3x2 factorial design.

6.4.2 Multiparameter tests

Whenever you are dealing with designs where all categorical factors have no more than two levels, the test of the regression coefficient associated with a given factor will be equivalent to the test for the effect in the ANOVA framework, provided you use sum or deviation coding. But in the above example, we have a 3x2 design, with two predictor variables coding the main effect of `meal` (`lunch_v_breakfast` and `dinner_v_breakfast`). Let's simulate some data and run a one-factor ANOVA with `aov()`, and then we'll replicate the analysis using the regression function `lm()` (note that the same procedure works for mixed-effects models on multilevel data, just using `lme4::lmer()` instead of `base::lm()`).

```
## make up some data
set.seed(62)
meals <- tibble(meal = factor(rep(c("breakfast", "lunch", "dinner"),
                                  each = 6)),
                time_of_week = factor(rep(rep(c("weekday", "weekend"),
                                              each = 3), 3)),
                calories = rnorm(18, 450, 50))

## use sum coding instead of default 'dummy' (treatment) coding
options(contrasts = c(unordered = "contr.sum", ordered = "contr.poly"))

aov(calories ~ meal * time_of_week, data = meals) %>%
  summary()
```

```
##                   Df Sum Sq Mean Sq F value Pr(>F)
## meal               2   2164    1082   0.380  0.692
## time_of_week       1   5084    5084   1.783  0.207
## meal:time_of_week  2   4767    2384   0.836  0.457
## Residuals         12  34209    2851
```

We get three F-tests, one for each main effect (`meal` and `time_of_week`) one for the interaction. What happens if we fit a model using `lm()`?

```
## add our own numeric predictors
meals2 <- meals %>%
  mutate(lunch_v_breakfast = if_else(meal == "lunch", 2/3, -1/3),
         dinner_v_breakfast = if_else(meal == "dinner", 2/3, -1/3),
         time_week = if_else(time_of_week == "weekend", 1/2, -1/2))

## double check our coding
distinct(meals2, meal, time_of_week,
         lunch_v_breakfast, dinner_v_breakfast, time_week)

## fit regression model
mod <- lm(calories ~ (lunch_v_breakfast + dinner_v_breakfast) *
            time_week, data = meals2)

summary(mod)
```

```
## # A tibble: 6 x 5
##   meal      time_of_week lunch_v_breakfast dinner_v_breakfast time_week
##   <fct>     <fct>                    <dbl>              <dbl>     <dbl>
## 1 breakfast weekday                 -0.333             -0.333      -0.5
## 2 breakfast weekend                 -0.333             -0.333       0.5
## 3 lunch     weekday                   0.667             -0.333      -0.5
## 4 lunch     weekend                   0.667             -0.333       0.5
## 5 dinner    weekday                  -0.333              0.667      -0.5
## 6 dinner    weekend                  -0.333              0.667       0.5
##
## Call:
## lm(formula = calories ~ (lunch_v_breakfast + dinner_v_breakfast) *
##     time_week, data = meals2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -68.522 -35.895  -4.063  42.061  73.081
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    451.15      12.58  35.848 1.42e-13 ***
## lunch_v_breakfast              -25.23      30.83  -0.818    0.429
## dinner_v_breakfast             -20.59      30.83  -0.668    0.517
## time_week                       33.61      25.17   1.335    0.207
## lunch_v_breakfast:time_week    -58.31      61.65  -0.946    0.363
## dinner_v_breakfast:time_week    17.93      61.65   0.291    0.776
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 53.39 on 12 degrees of freedom
## Multiple R-squared:  0.2599,Adjusted R-squared:  -0.04843
## F-statistic: 0.843 on 5 and 12 DF,  p-value: 0.5447
```

OK, this output looks very different! How do you perform ANOVA-like tests in those situations? You have estimates for `lunch_v_breakfast` and `dinner_v_breakfast` but how would you convert this into a single test for the main effect of `meal`? Likewise, you have two interaction terms, `lunch_v_breakfast:tow` and `dinner_v_breakfast:tow`; how do you convert this into a single test for the interaction?

The solution is to perform **multiparameter tests** using model comparison, implemented by the `anova()` function in R. To test for the main effect of `meal`, you'd compare a model containing the two predictor variables coding that factor (`lunch_v_breakfast` and `dinner_v_breakfast`) to a model excluding these two predictors but that is otherwise identical. You can either re-fit the model by writing it out and excluding the terms, or by using the `update()` function and removing the terms (which is a shortcut). Let's write it out first.

```
## fit the model
mod_main_eff <- lm(calories ~ time_week +
                     lunch_v_breakfast:time_week + dinner_v_breakfast:time_week,
                   meals2)
```

```
## compare models
anova(mod, mod_main_eff)
```

```
## Analysis of Variance Table
##
## Model 1: calories ~ (lunch_v_breakfast + dinner_v_breakfast) * time_week
## Model 2: calories ~ time_week + lunch_v_breakfast:time_week + dinner_v_breakfast:ti
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1     12 34209
## 2     14 36373 -2   -2163.9 0.3795 0.6921
```

OK, now here is an equivalent version using the shortcut `update()` function, which takes the model you want to update as the first argument and then a special syntax for the formula including your changes. For the formula, we use `. ~ . -lunch_v_breakfast -dinner_v_breakfast~` Although this formula seems weird, the dot `.` says "keep everything on this side of the model formula (left of `~`) as it is in the original model." So the formula `. ~ .` would use the same formula as the original model; that is, `update(mod, . ~ .)` would fit the exact same model as above. In contrast, `. ~ . -x -y` means "everything the same on the left side (same DV), but remove variables `x` and `y` from the right side.

```
mod_main_eff2 <- update(mod, . ~ . -lunch_v_breakfast -dinner_v_breakfast)

anova(mod, mod_main_eff2)
```

```
## Analysis of Variance Table
##
## Model 1: calories ~ (lunch_v_breakfast + dinner_v_breakfast) * time_week
## Model 2: calories ~ time_week + lunch_v_breakfast:time_week + dinner_v_breakfast:ti
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1     12 34209
## 2     14 36373 -2   -2163.9 0.3795 0.6921
```

As you can see, this gives us the same result as above.

If we want to test the main effect of `time_of_week`, we remove that predictor.

```
mod_tow <- update(mod, . ~ . -time_week)

anova(mod, mod_tow)
```

```
## Analysis of Variance Table
##
## Model 1: calories ~ (lunch_v_breakfast + dinner_v_breakfast) * time_week
## Model 2: calories ~ lunch_v_breakfast + dinner_v_breakfast + lunch_v_breakfast:time
##     dinner_v_breakfast:time_week
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1     12 34209
## 2     13 39294 -1   -5084.1 1.7834 0.2065
```

Try to figure out how to test the interaction on your own.

Click to see solution

```
mod_interact <- update(mod, . ~ . -lunch_v_breakfast:time_week
                       -dinner_v_breakfast:time_week)

anova(mod, mod_interact)
```

```
## Analysis of Variance Table
##
## Model 1: calories ~ (lunch_v_breakfast + dinner_v_breakfast) * time_week
## Model 2: calories ~ lunch_v_breakfast + dinner_v_breakfast + time_week
##   Res.Df   RSS Df Sum of Sq      F Pr(>F)
## 1     12 34209
## 2     14 38977 -2   -4767.4 0.8362 0.4571
```

We got the exact same results doing model comparison with `lm()` as we did using `aov()`. Although this involved more steps, it is worth learning this approach as it will ultimately give you much more flexibility.

Chapter 7

Linear mixed-effects models with crossed random factors

7.1 Learning objectives

- analyze data from a design with crossed random factors of subjects and stimuli
- appropriately specify random effects to enable proper generalization
- simulate data for a design with crossed random factors

7.2 Web app

- Demo of crossed random effects

7.3 Generalizing over encounters between subjects and stimuli

A common goal of experiments in psychology is to test claims about behavior that arises in response to certain types of stimuli (or sometimes, the neural underpinning of that behavior). The stimuli might be, for instance, words, images, sounds, videos, or stories. Some examples of claims you might want to test are:

- when listening to words in a second language, do bilinguals experience interference from words in their native language?
- do people rate the attractiveness of faces differently when they are in a good mood than when they are in a bad mood?

- does viewing soothing images help reduce stress relative to more neutral images?
- when reading a scenario ambiguously describing a target individual, are people more likely to make assumptions about what social group the target belongs to after being subliminally primed?

One thing to note about all these claims is that the are of the type, "what happens to our measurements when an individual of type X encounters a stimulus of type Y", where X is drawn from a target population of subjects and Y is drawn from a target population of stimuli. In other words, we are attempt to make generalizable claims about a particular class of **events** involving **encounters** between sampling units of subjects and stimuli (Barr, 2017). But just like we can't sample all possible subjects from the target population of subjects, we also cannot sample all possible stimuli from the target population of stimuli. Thus, when drawing inferences, we need to account for the uncertainty introduced in our estimates by *both* sampling processes (Coleman, 1964; Clark, 1973; Judd et al., 2012; Yarkoni, 2019). Linear mixed-effects models make it particularly easy to do this by allowing more than one random factor in our model formula (Baayen et al., 2008).

Here is a simple example of a study where you are interested in testing whether people rate pictures of cats, dogs, or sunsets as more soothing images to look at. You want to say something general about the category of cats, dogs, and sunsets and not something about the specific pictures that you happened to sample. Let's say you randomly select four images from each of the three categories from Google Images (you would absolutely need to have more to be able to say something generalizable, but we chose a small number to keep the example simple). So your table of stimuli might look like the following:

stimulus_id	category	file
1	cat	cat1.jpg
2	cat	cat2.jpg
3	cat	cat3.jpg
4	cat	cat4.jpg
5	dog	dog1.jpg
6	dog	dog2.jpg
7	dog	dog3.jpg
8	dog	dog4.jpg
9	sunset	sunset1.jpg
10	sunset	sunset2.jpg
11	sunset	sunset3.jpg
12	sunset	sunset4.jpg

Then you sample a set of four participants to perform the soothing ratings. Again, four would be too few for a real study, but we're keeping it small just for expository purposes.

subject_id	age	date
1	54	2020-05-02
2	45	2020-05-13
3	37	2020-05-21
4	49	2020-05-27

Now, because each subject has given a "soothingness" rating for each picture, you'd have a full dataset consisting of all of the levels of `subject_id` crossed with all of the levels of `stimulus_id`. This is what we mean when we talk about "crossed random factors." You can create the table containing all these combinations with the `crossing()` function from `tidyr` (which is loaded when you load in `tidyverse`).

```
crossing(subjects %>% select(subject_id),
         stimuli %>% select(-category))
```

subject_id	stimulus_id	file
1	1	cat1.jpg
1	2	cat2.jpg
1	3	cat3.jpg
1	4	cat4.jpg
1	5	dog1.jpg
1	6	dog2.jpg
1	7	dog3.jpg
1	8	dog4.jpg
1	9	sunset1.jpg
1	10	sunset2.jpg
1	11	sunset3.jpg
1	12	sunset4.jpg
2	1	cat1.jpg
2	2	cat2.jpg
2	3	cat3.jpg
2	4	cat4.jpg
2	5	dog1.jpg
2	6	dog2.jpg
2	7	dog3.jpg
2	8	dog4.jpg
2	9	sunset1.jpg
2	10	sunset2.jpg
2	11	sunset3.jpg
2	12	sunset4.jpg
3	1	cat1.jpg
3	2	cat2.jpg
3	3	cat3.jpg
3	4	cat4.jpg
3	5	dog1.jpg
3	6	dog2.jpg
3	7	dog3.jpg
3	8	dog4.jpg
3	9	sunset1.jpg
3	10	sunset2.jpg
3	11	sunset3.jpg
3	12	sunset4.jpg
4	1	cat1.jpg
4	2	cat2.jpg
4	3	cat3.jpg
4	4	cat4.jpg
4	5	dog1.jpg
4	6	dog2.jpg
4	7	dog3.jpg
4	8	dog4.jpg
4	9	sunset1.jpg
4	10	sunset2.jpg
4	11	sunset3.jpg
4	12	sunset4.jpg

Because you have 4 subjects responding to 12 stimuli, the resulting table will have 48 rows.

7.4 lme4 syntax for crossed random factors

How should we analyze such data? Recall from the last chapter that the lme4 formula syntax for a model with by-subject random intercepts and slopes for predictor `x` would be given by `y ~ x + (1 + x | subject_id)` where the term in brackets with the vertical bar `|` provides the random effects specification. The variable to the right of the bar, `subject_id`, specifies the variable that identifies the levels of the random factor. The formula to the left of the bar within the brackets, `1 + x`, specifies the random effects associated with this factor, which in this case is a random intercept and random slope for `x`. The best way to think about this bracketed part of the formula `(1 + x | subject_id)` is **as an instruction to `lme4::lmer()` about how to build a covariance matrix capturing the variance introduced by the random factor of subjects.** By now you should realize that this instruction would result in the estimation of a two-dimensional covariance matrix, with one dimension for intercept variance and one for slope variance.

But we are not limited to the estimation of random effects for subjects; we can also specify the estimation of random effects for stimuli by simply adding another term to the formula. For example,

```
y ~ x + (1 + x | subject_id) + (1 + x | stimulus_id)
```

regresses `y` on `x` with by-subject random intercepts and slopes and by-stimulus random intercepts. In this way, the fitted model will capture two sources of uncertainty about our estimates—the uncertainty introduced by sampling subjects as well as the uncertainty introduced by sampling items. Now we are estimating **two independent covariance matrices**, one for subjects and one for items. In the above example, both of these matrices will have the same 2x2 structure, but this need not be the case. We can flexibly change the random effects structure by changing the formula specification on the left side of each bar `|` symbol. For instance, if we have another predictor `w`, we might have:

```
y ~ x + (x | subject_id) + (x + w | stimulus_id)
```

which would estimate the same 2x2 matrix for subjects, but the covariance matrix for stimuli would now be a 3x3 matrix (intercepts, slope of x, and slope of w). Although this enables great flexibility, as the random effects structure becomes more complex, the estimation process becomes more difficult and less likely to converge upon a result.

7.5 Specifying random effects

The choice of random effects structure is not a new problem that appeared with linear mixed-effects models. In traditional approaches using t-test and ANOVA, you choose the random effects structure implicitly when you choose what procedure to use. As discussed in the last chapter, if you choose a paired samples t-test over an independent samples t-test, that is analogous to choosing to fit `lme4::lmer(y ~ x + (1 | subject))` over `lm(y ~ x)`. Likewise, you can run a mixed model ANOVA as either `aov(y ~ x + Error(subject_id))`, which is equivalent to a random intercepts model, or `aov(y ~ x + Error(x / subject_id))`, which is equivalent to a random slopes model. The tradition in psychology when performing confirmatory analyses is to use the *maximal random effects structure justified by the design of your study.* So, if you have one-factor data with pseudoreplications, you would use `aov(y ~ x + Error(x / subject_id))` rather than `aov(y ~ x + Error(subject_id))`. Analogously, if you were to analyze the same data with pseudoreplications using a linear mixed effects model, you should use `lme4::lmer(y ~ x + (1 + x | subject_id))` rather than `lme4::lmer(y ~ x + (1 | subject_id))`. In other words, you should account for all sources of non-independence introduced by repeated sampling from the same subjects or stimuli. This approach is known as the **maximal random effects** approach or the **design-driven approach** to specifying random effects structure (Barr et al., 2013). Failing to account for dependencies introduced by the design is likely to lead to standard errors that are too small, which in turn, lead to p-values that are smaller than they should be, and thus, higher false positive (Type I error) rates. In some cases, it can lead to lower power, and thus a higher false negative (Type II error) rate. It is thus of critical importance to pay close attention to the random effects structure when performing analyses.

Linear mixed-effects models almost inevitably include random intercepts for any random factor included in the design. So if your random factors are subjects and stimuli identified by `subject_id` and `stimulus_id` respectively, then at the very least, your model syntax will include `(1 | subject_id) + (1 | stimulus_id)`. But you will have various predictors in the model, so key question becomes: what predictors should I allow to vary over what sampling units? For instance, if the fixed-effects part of your model is a 2x2 factorial design with factors A and B, `y ~ a * b + ...`, you could have a large variety of random effects structures, including (but not limited to):

1. random intercepts only: `y ~ a * b + (1 | subject_id) + (1 | stimulus_id)`
2. by-subjects random intercepts for a and by-stimulus random intercepts: `y ~ a * b + (a | subject_id) + (1 | stimulus_id)`
3. by-subjects random intercepts and slopes for a and b and the ab interaction, with by-stimulus random intercepts: `y ~ a * b + (a * b | subject_id) + (1 | stimulus_id)`
4. by-subjects random intercepts and slopes for a and b and the ab

interaction, by-stimulus random intercepts and slopes for a and b and the ab interaction, `y ~ a * b + (a * b | subject_id) + (a * b | stimulus_id)`.

It is important to be clear about one thing.

The "maximal random effects structure justified by the design" is not the same as the "maximum possible random effects structure"; that is, it does not entail automatically putting in all random slopes for all random factors for all predictors in your model. You have to follow the guidelines for random effects in the next section to decide whether inclusion of a particular random slope is, in fact, "justified by the design."

Some authors suggest a "data-driven" alternative to design-driven random effects, suggesting that researchers should only include random slopes justified by the design if they are further justified by the data (Matuschek et al., 2017). For example, you might use a null-hypothesis test to determine whether including a by-subject random slope for `x` significantly improves the model fit, and only include that effect if it does. Although this could potentially improve power for the test of theoretical interest when random slopes are very small, it also exposes you to additional unknown risk of false positives, so it is questionable whether this the right approach in a confirmatory context. Thus, we do not recommend a data-driven approach.

7.5.1 Rules for choosing random effects for categorical factors

The random effects structure for a linear mixed-effects model—in other words, your assumptions about what effects vary over what sampling units—is absolutely critical for ensuring that your parameters reflect the uncertainty introduced by sampling (Barr et al., 2013). First off, note that we are focused on predictors representing **design variables** that are of theoretical interest and on which you will perform inferential tests. If you have predictors that represent **control variables**, over which you do not intend to perform statistical tests, it is unlikely that random slopes are needed.

The following rules are derived from Barr et al. (2013) and Barr (2013). Consult these papers if you wish to find out more about these guidelines. Keep in mind that you can only use a mixed effects model if you have repeated measures data, either because of pseudoreplications and/or the presence of within-subject (or within-stimulus) factors. With crossed random factors, you inevitably have pseudoreplications—multiple observations per subject due to multiple stimuli, multiple observations per stimulus due to multiple subjects. The key to determining random effects structure is figuring out which factors are within-subjects or within-stimuli, and where any pseudoreplications are located in the design. You apply the rules once for subjects to determine the form of the `(1 + ... | subject_id)` part of the formula, and once for stimuli to determine the form of the `(1 + ... | stimulus_id)` part of the formula. Where you see the word "unit" or "sampling unit" below, substitute "subject" or "stimuli" as needed.

Here are the rules:

1. If there are repeated measures on sampling units, you need a random intercept for that random factor: `(1 | unit_id)`;
2. If a factor `x` is between-unit, you do not need a random slope for that factor;
3. Determine the highest order interaction of within-subject factors for the unit under consideration. If you have pseudoreplications within each cell defined by those combinations (i.e., multiple observations per cell), then for that unit you will need a slope for that interaction as well as for all lower order effects. If there are no pseudoreplications, then you do not need any random slopes.

The first two rules are straightforward, but the third requires some explanation. Let's first ask: how do we know whether some factor is between or within unit?

A simple way to determine whether a factor is between or within is to use the `dplyr::count()` function, which gives frequency counts, and which is loaded when you load tidyverse. Let's say you are interested in whether factor A is within or between subjects, for the imaginary 2x2x2 factorial data `abc_data` below where A, B, and C name the factors of your design.

```
library("tidyverse")

## run this code to create the table "abc_data"
abc_subj <- tibble(subject_id = 1:4,
                   B = rep(c("B1", "B2"), times = 2))

abc_item  <- tibble(stimulus_id = 1:4,
                    A = rep(c("A1", "A2"), each = 2),
                    C = rep(c("C1", "C2"), times = 2))

abc_data <- crossing(abc_subj, abc_item) %>%
  select(subject_id, stimulus_id, everything())
```

To see whether A is within or between subjects, use:

```
abc_data %>%
  count(subject_id, A)
```

```
## # A tibble: 8 x 3
##   subject_id A         n
##        <int> <chr> <int>
## 1          1 A1        2
## 2          1 A2        2
## 3          2 A1        2
## 4           2 A2        2
## 5           3 A1        2
## 6           3 A2        2
## 7           4 A1        2
## 8           4 A2        2
```

In the resulting table, you can see that each subject gets both levels of A, making it a within-subject factor. What about B and C?

```
abc_data %>%
  count(subject_id, B)
```

```
## # A tibble: 4 x 3
##   subject_id B         n
##        <int> <chr> <int>
## 1          1 B1        4
## 2          2 B2        4
## 3          3 B1        4
## 4          4 B2        4
```

```
abc_data %>%
  count(subject_id, C)
```

```
## # A tibble: 8 x 3
##   subject_id C         n
##        <int> <chr> <int>
## 1          1 C1        2
## 2          1 C2        2
## 3          2 C1        2
## 4          2 C2        2
## 5          3 C1        2
## 6          3 C2        2
## 7          4 C1        2
## 8          4 C2        2
```

OK B is between subjects (each subject gets only one level), and C is within (each subject gets all levels).

Exercise

Answer these question about `abc_data`.

- Are the levels of factor A administered between or within stimuli? between within

Solution

```
abc_data %>%
  count(stimulus_id, A)
```

```
## # A tibble: 4 x 3
##   stimulus_id A         n
##         <int> <chr> <int>
## 1           1 A1        4
## 2           2 A1        4
## 3           3 A2        4
## 4           4 A2        4
```

- Are the levels of factor B administered between or within stimuli? between within

Solution

```
abc_data %>%
  count(stimulus_id, B)
```

```
## # A tibble: 8 x 3
##   stimulus_id B         n
##         <int> <chr> <int>
## 1           1 B1        2
## 2           1 B2        2
## 3           2 B1        2
## 4           2 B2        2
## 5           3 B1        2
## 6           3 B2        2
## 7           4 B1        2
## 8           4 B2        2
```

- Are the levels of factor C administered between or within stimuli? between within

Solution

```
abc_data %>%
  count(stimulus_id, C)
```

```
## # A tibble: 4 x 3
##   stimulus_id C         n
##         <int> <chr> <int>
## 1           1 C1        4
## 2           2 C2        4
## 3           3 C1        4
## 4           4 C2        4
```

OK, we've identified which factors are within and between subject, and which factors are within and between stimulus.

The second rule tells us that if a factor is between-unit, you do not need a random slope for that factor. Indeed, it is not possible to estimate a random slope for a between unit factor. If you think about the fact that random slopes capture variation in the effect over units, then it makes sense that you have to measure your response variable across all levels of that factor to be able to estimate that variation. For instance, if you have a two-level factor called treatment group (experimental, control), you cannot estimate the effect of "treatment" for a particular subject unless the subject has experienced both levels of the factor (i.e., it would have to be within-subject).

How do we now apply the third rule above to determine what random slopes are needed for our within-unit factors?

Consider that A and C were within-subjects, and B was between. So the highest-order interaction of within-subject factors is AC. We will need random slopes for the AC interaction as well as for the main effects A and C if we have pseudoreplications for each subject in each combination of AC. How can we find out?

```
abc_data %>%
  count(subject_id, A, C)
```

```
## # A tibble: 16 x 4
##    subject_id A     C         n
##         <int> <chr> <chr> <int>
##  1          1 A1    C1        1
##  2          1 A1    C2        1
##  3          1 A2    C1        1
##  4          1 A2    C2        1
##  5          2 A1    C1        1
##  6          2 A1    C2        1
##  7          2 A2    C1        1
##  8          2 A2    C2        1
##  9          3 A1    C1        1
## 10          3 A1    C2        1
## 11          3 A2    C1        1
## 12          3 A2    C2        1
## 13          4 A1    C1        1
## 14          4 A1    C2        1
## 15          4 A2    C1        1
## 16          4 A2    C2        1
```

This shows us that we have *one* observation per combination of AC, so we do not need random slopes for AC, nor for A or C. The random effects part of the formula for subjects would just be `(1 | subject_id)`.

What random slopes do you need for the random factor of stimulus?

Solution

You have one within-stimulus factor, B, which has pseudoreplications.

```
abc_data %>%
  count(stimulus_id, B)
```

```
## # A tibble: 8 x 3
##   stimulus_id B         n
##         <int> <chr> <int>
## 1           1 B1        2
## 2           1 B2        2
## 3           2 B1        2
## 4           2 B2        2
## 5           3 B1        2
## 6           3 B2        2
## 7           4 B1        2
## 8           4 B2        2
```

Therefore the formula you need for stimuli is `(B | stimulus_id)`, making the full `lme4` formula:

`y ~ A * B * C + (1 | subject_id) + (B | stimulus_id)`.

7.6 Simulating data with crossed random factors

For these exercises, we will generate simulated data corresponding to an experiment with a single, two-level factor (independent variable) that is within-subjects and between-items. Let's imagine that the experiment involves lexical decisions to a set of words (e.g., is "PINT" a word or nonword?), and the dependent variable is response time (in milliseconds), and the independent variable is word type (noun vs verb). We want to treat both subjects and words as random factors (so that we can generalize to the population of events where subjects encounter words). You can play around with the web app (or click here to open it in a new window), which allows you to manipulate the data-generating parameters and see their effect on the data.

By now, you should have all the pieces of the puzzle that you need to simulate data from a study with crossed random effects. DeBruine and Barr (2020) provides a more detailed, step-by-step walkthrough of the exercise below.

Here is the DGP for response time Y_{si} for subject s and item i:

Level 1:

$$Y_{si} = \beta_{0s} + \beta_1 X_i + e_{si} \tag{7.1}$$

Level 2:

$$\beta_{0s} = \gamma_{00} + S_{0s} + I_{0i} \tag{7.2}$$

$$\beta_1 = \gamma_{10} + S_{1s} \tag{7.3}$$

Variance Components:

$$\langle S_{0s}, S_{1s} \rangle \sim N\left(\langle 0, 0 \rangle, \ \right) \tag{7.4}$$

$$= \left(\begin{array}{cc} {\tau_{00}}^2 & \rho\tau_{00}\tau_{11} \\ \rho\tau_{00}\tau_{11} & {\tau_{11}}^2 \end{array} \right) \tag{7.5}$$

$$I_{0s} \sim N\left(0, {\omega_{00}}^2\right) \tag{7.6}$$

$$e_{si} \sim N\left(0, \sigma^2\right) \tag{7.7}$$

In the above equation, X_i is a numerical predictor coding which condition the item i is in; e.g., -.5 for noun, .5 for verb.

We could just reduce levels 1 and 2 to

$$Y_{si} = \beta_0 + S_{0s} + I_{0i} + (\beta_1 + S_{1s})X_i + e_{si}$$

where:

Parameter	Symbol	Description
Y_{si}	`Y`	RT for subject s responding to item i;
β_0	`b0`	grand mean;
S_{0s}	`S_0s`	random intercept for subject s s;
I_{0i}	`I_0i`	random intercept for item i i;
β_1	`b1`	fixed effect of word type (slope);
S_{1s}	`S_1s`	by-subject random slope;
X_i	`cond`	deviation-coded predictor variable for word type;
τ_{00}	`tau_00`	by-subject random intercept standard deviation
τ_{11}	`tau_11`	by-subject random slope standard deviation
ρ	`rho`	correlation between random intercept and slope
ω_{00}	`omega_00`	by-item random intercept standard deviation
e_{si}	`err`	residual for subject s item i error
σ	`sig`	residual error standard deviation

7.6.1 Set up the environment and define the parameters for the DGP

If you want to get the same results as everyone else for this exercise, then we all should seed the random number generator with the same value. While we're at it, let's load in the packages we need.

```r
library("lme4")
library("tidyverse")

set.seed(11709)
```

Now let's define the parameters for the DGP (data generating process).

```r
nsubj <- 100 # number of subjects
nitem <- 50  # must be an even number

b0 <- 800 # grand mean
b1 <- 80 # 80 ms difference
effc <- c(-.5, .5) # deviation codes

omega_00 <- 80 # by-item random intercept sd (omega_00)

## for the by-subjects variance-covariance matrix
tau_00 <- 100 # by-subject random intercept sd
tau_11 <- 40 # by-subject random slope sd
rho <- .2 # correlation between intercept and slope

sig <- 200 # residual (standard deviation)
```

You'll create three tables:

Name	Description
`subjects`	table of subject data including `subj_id` and subject random effects
`items`	table of stimulus data including `item_id` and item random effect
`trials`	table of trials enumerating encounters between subjects/stimuli

Then you will merge together the information in the three tables, and calculate the response variable according to the model formula above.

7.6.2 Generate a sample of stimuli

Let's randomly generate our 50 items. Create a tibble called `item` like the one below, where `iri` are the by-item random intercepts (drawn from a normal distribution with variance $\omega_{00}^2 = 6400$). Half of the words are of type NOUN (`cond = -.5`) and half of type VERB (`cond = .5`).

Click to reveal full table

```
## # A tibble: 50 x 3
##    item_id  cond    I_0i
##       <int> <dbl>   <dbl>
##  1        1  -0.5   14.9
##  2        2   0.5  -86.3
##  3        3  -0.5  -12.8
##  4        4   0.5  -13.9
##  5        5  -0.5   55.6
##  6        6   0.5  -45.9
##  7        7  -0.5  -42.0
##  8        8   0.5  -87.6
##  9        9  -0.5  -97.4
## 10       10   0.5  -85.2
## 11       11  -0.5  135.
## 12       12   0.5   83.2
## 13       13  -0.5  -44.7
## 14       14   0.5    8.59
## 15       15  -0.5 -156.
## 16       16   0.5  -57.6
## 17       17  -0.5  -38.7
## 18       18   0.5   39.6
## 19       19  -0.5  105.
## 20       20   0.5   30.3
## 21       21  -0.5 -115.
## 22       22   0.5   -3.40
## 23       23  -0.5 -218.
## 24       24   0.5   53.0
## 25       25  -0.5  -86.9
## 26       26   0.5  -65.4
## 27       27  -0.5  172.
## 28       28   0.5 -152.
## 29       29  -0.5   25.1
## 30       30   0.5 -156.
## 31       31  -0.5   47.7
## 32       32   0.5  -46.3
## 33       33  -0.5   48.0
## 34       34   0.5   62.8
## 35       35  -0.5  -75.4
## 36       36   0.5  -35.9
## 37       37  -0.5  -48.5
## 38       38   0.5   29.3
## 39       39  -0.5  -55.5
## 40       40   0.5   69.5
## 41       41  -0.5  196.
## 42       42   0.5   77.6
## 43       43  -0.5  -45.0
## 44       44   0.5  204.
## 45       45  -0.5   32.1
## 46      46   0.5  -63.9
## 47      47  -0.5  145.
## 48      48   0.5   66.2
## 49      49  -0.5  -23.9
## 50      50   0.5   97.3
```

Hint for making cond

`rep()`

Hint for making item random effects

`rnorm()`

Solution

```
items <- tibble(item_id = 1:nitem,
                cond = rep(c(-.5, .5), times = nitem / 2),
                I_0i = rnorm(nitem, 0, sd = omega_00))
```

7.6.3 Generate a sample of subjects

To generate the by-subject random effects, you will need to generate data from a *bivariate normal distribution*. To do this, we will use the function `MASS::mvrnorm`.

REMEMBER: do not run `library("MASS")` just to get this one function, because `MASS` has a function `select()` that will overwrite the tidyverse version. Since all we want from MASS is the `mvrnorm()` function, we can just access it directly by the `pkgname::function` syntax, i.e., `MASS::mvrnorm()`.

Your subjects table should look like this:

Click to reveal full table

```
## # A tibble: 100 x 3
##     subj_id      S_0s     S_1s
##       <int>     <dbl>    <dbl>
##   1       1  -80.0      -0.763
##   2       2   44.6      54.5
##   3       3    8.74    -20.4
##   4       4  -38.6     -23.8
##   5       5  -83.3      29.2
##   6       6  -70.9     -13.8
##   7       7  -21.4      46.0
##   8       8    2.33      8.39
##   9       9   62.3     -58.2
##  10      10  238.        7.72
##  11      11  -92.5       2.14
##  12      12   58.5     -65.8
## 13       13 -204.       -38.8
## 14       14  -91.6        5.46
## 15       15   51.1      -38.8
## 16       16  142.       -12.9
## 17       17   46.0        6.60
## 18       18  -56.7      -54.8
## 19       19  -10.1       62.1
## 20       20 -226.       -19.3
## 21       21 -158.       -18.5
## 22       22  102.         8.99
## 23       23  -12.7      -70.6
## 24       24  135.        -9.50
## 25       25   62.0      -52.5
## 26       26    0.0653   32.8
## 27       27 -117.        70.8
## 28       28 -232.         3.43
## 29       29   70.9       50.8
## 30       30 -123.        22.8
## 31       31  268.        30.0
## 32       32  -18.7      -25.0
## 33       33   50.8      -31.0
## 34       34  -43.1      -28.9
## 35       35  -10.1       28.3
## 36       36   65.6       18.2
## 37       37 -123.        -4.63
## 38       38  -94.8       10.3
## 39       39   77.7      -22.5
## 40       40  -59.1       52.4
## 41       41  -91.2     -103.
## 42       42  -66.6       -2.14
## 43       43   -4.40       0.305
## 44       44   69.7       10.2
## 45       45  -77.5      -10.4
## 46       46  -17.8      -48.2
## 47       47 -103.        47.0
## 48       48   22.8      -39.3
## 49       49  -31.1      -34.9
## 50       50  -26.4       40.0
## 51       51   47.8       26.0
## 52       52  -93.2      -42.7
## 53       53   28.9       51.4
## 54       54  -19.3       11.5
## 55       55   53.6       21.5
## 56       56  -27.4      -21.4
## 57       57  -67.7      -32.1
## 58       58   59.2       13.4
## 59      59  -53.1        2.44
## 60      60  104.         7.41
## 61      61  -20.7      -78.7
## 62      62   55.9      -15.7
## 63      63  114.       -29.1
## 64      64  -57.7      -34.7
## 65      65  -38.7       -9.14
## 66      66 -106.       -58.0
## 67      67   99.1      -37.6
## 68      68  -56.9       21.0
## 69      69  -50.4       -0.407
## 70      70   27.5       -2.69
## 71      71  139.       -32.2
## 72      72   44.9        8.53
## 73      73  -14.8       71.7
## 74      74   33.7      -52.6
## 75      75    2.03      27.8
## 76      76 -134.        37.0
## 77      77   24.4       20.7
## 78      78  -60.6      -36.7
## 79      79   31.1       16.9
## 80      80  -34.9        9.68
## 81      81  206.        17.3
## 82      82   -7.19     -25.4
## 83      83  182.        46.0
## 84      84   55.7       21.7
## 85      85 -149.       -44.0
## 86      86 -193.       -73.2
## 87      87  167.        13.9
## 88      88  160.         3.87
## 89      89   84.1       82.1
## 90      90   97.2       -6.55
## 91      91 -205.      -125.
## 92      92  -75.1        6.76
## 93      93  -95.3      -46.5
## 94      94  106.        38.6
## 95      95  -42.4       11.3
## 96      96   74.0      -21.1
## 97      97 -245.       -25.3
## 98      98 -113.        -1.88
## 99      99   68.8       30.6
## 100    100  136.        44.2
```

Hint 1

recall that:

- *tau_00*: by-subject random intercept standard deviation
- *tau_11*: by-subject random slope standard deviation
- *rho* : correlation between intercept and slope

Hint 2

`covariance = rho * tau_00 * tau_11`

Hint 3

```
matrix(    tau_00^2,            rho * tau_00 * tau_11,
        rho * tau_00 * tau_11,      tau_11^2, ...)
```

Hint 4

```
as_tibble(mx) %>%
  mutate(subj_id = ...)
```

Solution

```
cov <- rho * tau_00 * tau_11

mx <- matrix(c(tau_00^2, cov,
               cov,      tau_11^2),
             nrow = 2)

by_subj_rfx <- MASS::mvrnorm(nsubj, mu = c(S_0s = 0, S_1s = 0), Sigma = mx)

subjects <- as_tibble(by_subj_rfx) %>%
  mutate(subj_id = row_number()) %>%
  select(subj_id, everything())
```

7.6.4 Generate a sample of encounters (trials)

Each trial is an *encounter* between a particular subject and stimulus. In this experiment, each subject will see each stimulus. Generate a table `trials` that lists the encounters in the experiments. Note: each participant encounters each stimulus item once. Use the `crossing()` function to create all possible encounters.

Now apply this example to generate the table below, where `err` is the residual term, drawn from $N \sim (0, \sigma^2)$, where σ is `err_sd`.

```
## # A tibble: 5,000 x 3
##    subj_id item_id    err
##      <int>   <int>  <dbl>
##  1       1       1  382.
##  2       1       2  283.
##  3       1       3   30.4
##  4       1       4 -282.
##  5       1       5 -239.
##  6       1       6   73.4
##  7       1       7  -98.4
##  8       1       8 -189.
##  9       1       9 -410.
## 10       1      10  102.
## # ... with 4,990 more rows
```

Solution

```
trials <- crossing(subj_id = subjects %>% pull(subj_id),
                   item_id = items %>% pull(item_id)) %>%
  mutate(err = rnorm(nrow(.), mean = 0, sd = sig))
```

7.6.5 Join subjects, items, and trials

Merge the information in `subjects`, `items`, and `trials` to create the full dataset `dat_sim`, which looks like this:

```
## # A tibble: 5,000 x 7
##    subj_id item_id  S_0s   I_0i   S_1s  cond    err
##      <int>   <int> <dbl>  <dbl>  <dbl> <dbl>  <dbl>
##  1       1       1 -80.0  14.9 -0.763  -0.5  382.
##  2       1       2 -80.0 -86.3 -0.763   0.5  283.
##  3       1       3 -80.0 -12.8 -0.763  -0.5   30.4
##  4       1       4 -80.0 -13.9 -0.763   0.5 -282.
##  5       1       5 -80.0  55.6 -0.763  -0.5 -239.
##  6       1       6 -80.0 -45.9 -0.763   0.5   73.4
##  7       1       7 -80.0 -42.0 -0.763  -0.5  -98.4
##  8       1       8 -80.0 -87.6 -0.763   0.5 -189.
##  9       1       9 -80.0 -97.4 -0.763  -0.5 -410.
## 10       1      10 -80.0 -85.2 -0.763   0.5  102.
## # ... with 4,990 more rows
```

Hint

`inner_join()`

Solution

```
dat_sim <- subjects %>%
  inner_join(trials, "subj_id") %>%
  inner_join(items, "item_id") %>%
  arrange(subj_id, item_id) %>%
  select(subj_id, item_id, S_0s, I_0i, S_1s, cond, err)
```

7.6.6 Create the response variable

Add the response variable `Y` to dat according to the model formula:

$$Y_{si} = \beta_0 + S_{0s} + I_{0i} + (\beta_1 + S_{1s})X_i + e_{si}$$

so that the resulting table (`dat_sim2`) looks like this:

```
## # A tibble: 5,000 x 8
##    subj_id item_id     Y  S_0s  I_0i   S_1s  cond    err
##      <int>   <int> <dbl> <dbl> <dbl>  <dbl> <dbl>  <dbl>
##  1       1       1 1078. -80.0  14.9 -0.763  -0.5  382.
##  2       1       2  957. -80.0 -86.3 -0.763   0.5  283.
##  3       1       3  698. -80.0 -12.8 -0.763  -0.5   30.4
##  4       1       4  464. -80.0 -13.9 -0.763   0.5 -282.
##  5       1       5  497. -80.0  55.6 -0.763  -0.5 -239.
##  6       1       6  787. -80.0 -45.9 -0.763   0.5   73.4
##  7       1       7  540. -80.0 -42.0 -0.763  -0.5  -98.4
##  8       1       8  483. -80.0 -87.6 -0.763   0.5 -189.
##  9       1       9  173. -80.0 -97.4 -0.763  -0.5 -410.
## 10       1      10  776. -80.0 -85.2 -0.763   0.5  102.
## # ... with 4,990 more rows
```

Note: this is the full **decomposition table** for this model.

Hint

```
... %>%
  mutate(Y = ...) %>%
  select(...)
```

Solution

```
dat_sim2 <- dat_sim %>%
  mutate(Y = b0 + S_0s + I_0i + (S_1s + b1) * cond + err) %>%
  select(subj_id, item_id, Y, everything())
```

7.6.7 Fitting the model

Now that you have created simulated data, estimate the model using `lme4::lmer()`, and run `summary()`.

Solution

```
mod_sim <- lmer(Y ~ cond + (1 + cond | subj_id) + (1 | item_id),
                dat_sim2, REML = FALSE)

summary(mod_sim, corr = FALSE)
```

```
## Linear mixed model fit by maximum likelihood  ['lmerMod']
## Formula: Y ~ cond + (1 + cond | subj_id) + (1 | item_id)
##    Data: dat_sim2
##
```

```
##      AIC      BIC   logLik deviance df.resid
##  67639.4  67685.0 -33812.7  67625.4     4993
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.6357 -0.6599 -0.0251  0.6767  3.7685
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  subj_id  (Intercept)  9464.8   97.29
##           cond          597.7   24.45   0.68
##  item_id  (Intercept)  8087.0   89.93
##  Residual             40305.0  200.76
## Number of obs: 5000, groups:  subj_id, 100; item_id, 50
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)   793.29      16.26  48.782
## cond           77.65      26.18   2.967
```

Now see if you can identify the data generating parameters in the output of `summary()`.

First, try to find β_0 and β_1.

Solution: Fixed effects

parameter	variable	input	estimate
\(\hat{\beta}_0\)	‘b0‘	800	793.293
\(\hat{\beta}_1\)	‘b1‘	80	77.652

Now try to find estimates of random effects parameters τ_{00}, τ_{11}, ρ, ω_{00}, and σ.

Solution: Random effects parameters

parameter	variable	input	estimate
\(\hat{\tau}_{00}\)	‘tau_00‘	100.0	97.287
\(\hat{\tau}_{11}\)	‘tau_11‘	40.0	24.448
\(\hat{\rho}\)	‘rho‘	0.2	0.675
\(\hat{\omega}_{00}\)	‘omega_00‘	80.0	89.928
\(\hat{\sigma}\)	‘sig‘	200.0	200.761

Chapter 8

Generalized linear mixed-effects models

8.1 Learning objectives

- distinguish discrete from continuous data
- estimate logit models

8.2 Discrete versus continuous data

All of the models we have been considering up to this point have assumed that the response (i.e. dependent) variable we are measuring is **continuous** and numeric. However, there are many cases in psychology where our measurements are of a **discrete** nature. By discrete, we mean that there are gaps between values, such as we would get with Likert scale data. It may also be the case that the values of the response variable reflect categories with no intrinsic ordering. Here are some examples:

- type of linguistic structure a speaker produces (double object or prepositional object phrase)
- which of a set of images a participant is viewing at a given moment
- whether the participant has made an accurate or inaccurate selection
- whether a job candidate gets hired or not
- agreement rating on a Likert scale.

Another common type of data is **count** data, where values are also discrete. Often with count data, the number of opportunities for something to occur is not well-defined. Some examples:

- number of speech errors in a corpus

- number of turn shifts between speakers in a conversation
- number of visits to the doctor in a given month.

8.2.1 Why not model discrete data as continuous?

Discrete data has some properties that generally make it a bad idea to try to analyze it using models intended for continuous data. For instance, if you are interested in the probability of some binary event (a participant's accuracy in a forced-choice task), each measurement will be represented as a 0 or a 1, indicating an inaccurate or an accurate response, respectively. You could calculate the proportion of accurate responses for each participant and analyze that (and many people do) but that is a bad idea for a number of reasons.

8.2.1.1 Bounded scale

Discrete data generally has a bounded scale. It may be bounded below (as with count data, where the lower bound is zero) or it may have both an upper and lower bound, such as Likert scale data. or binary data.

8.2.1.2 The variance is proportional to the mean

In most settings with continuous data, the variance is assumed to be independent of the mean; this is essentially the assumption of homogeneity of variance in a model with a continuous predictor. For discrete data, this assumption of the independence of the mean from the variance is often not met.

We can see this through data simulation. The `rbinom()` function makes it possible to simulate data from a **binomial** distribution, which describes how a collection of discrete observations behaves. Let's consider, for instance, the probability of rain on a given day in Barcelona, Spain, versus Glasgow, U.K. According to this website, Barcelona gets an average of 55 days of rain per year, while Glasgow gets 170. So the probability of rain on a given day in Glasgow can be estimated as 170/365 or about 0.47, where as the probability for Barcelona is 55/365 or about 0.15. Let's simulate 500 years of rainfall for the two cities (assumimg the climate remains constant).

```r
rainy_days <- tibble(city = rep(c("Barcelona", "Glasgow"), each = 500),
       days_of_rain = c(rbinom(500, 365, 55/365),
                        rbinom(500, 365, 170/365)))

rainy_days %>%
  ggplot(aes(days_of_rain)) +
  geom_histogram() +
  facet_wrap(~ city)
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

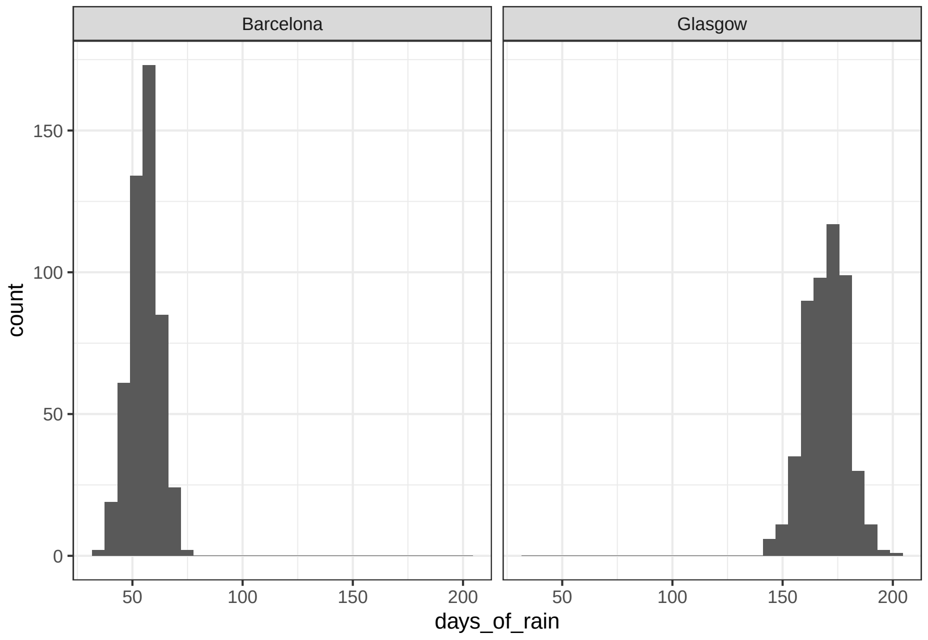

Figure 8.1: **Distribution of number of rainy days for 500 years of simulated rainfall in Barcelona and Glasgow**

The distribution for Glasgow is slightly fatter than the distribution for Barcelona. We can also see the greater variability in Glasgow if we look at the standard deviations of these variables.

```
rainy_days %>%
  group_by(city) %>%
  summarise(sd = sd(days_of_rain))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
## # A tibble: 2 x 2
##   city         sd
##   <chr>     <dbl>
## 1 Barcelona  6.72
## 2 Glasgow    9.30
```

8.2.1.3 Spurious interactions due to scaling effects

Another reason why treating discrete data as continuous can be problematic is the bounded nature of many discrete scales, which can lead to the detection of spurious interaction effects. For instance, consider the effect of some experimental intervention that increases accuracy. If participants are already highly accurate (e.g., more than 90%) in condition A than in condition B (say, 50%)

then the size of the possible effect in A is smaller than the size of the possible effect in B, since accuracy cannot exceed 100%. Thus, it is difficult to know whether any interaction effect reflects something theoretically meaningful, or just an artifact of the bounded nature of the scale.

8.3 Generalized Linear Models

The basic idea behind Generalized Linear Models (not to be confused with General Linear Models) is to specify a **link function** that transforms the response space into a modeling space where we can perform our usual linear regression, and to capture the dependence of the variance on the mean through a **variance function**. The parameters of the model will be expressed on the scale of the modeling space, but we can always transform it back into our original response space using the **inverse link function**.

There are a large variety of different kinds of generalized linear models you can fit to different types of data. The ones most commonly used in psychology are **logistic regression** and **Poisson regression**, the former being used for binary data (Bernoulli trials) and the latter being used for count data, where the number of trials is not well-defined. We will be focusing on logistic regression.

8.4 Logistic regression

8.4.1 Terminology

Term	Definition
Bernoulli trial	An event with a binary outcome, with one outcome considered 'success'
proportion	The ratio of successes to the total number of Bernoulli trials
odds	The ratio of successes to failures
log odds	The (natural) log of the odds

In logistic regression, we are modeling the relationship between the response and a set of predictors in log odds space.

Logistic regression is used when the individual outcomes are Bernoulli trials—events with binary outcomes. Typically one of the two outcomes is referred to as 'success' and is coded as a 1; the other is referred to as 'failure' and is coded as 0. Note that the terms 'success' and 'failure' are completely arbitrary, and should not be taken to imply that the more desireable category should always be coded as 1. For instance, when flipping a coin we could equivalently choose 'heads' as success and 'tails' as failure or vice-versa.

Often the outcome of a sequence of Bernoulli trials is communicated as a **proportion**—the ratio of successes to the total number of trials. For instance, if we flip a coin 100 times and get 47 heads, we would have a proportion of 47/100 or .47, which would also be our estimate of the probability of the event. For events coded as 1s and 0s, a shortcut way of getting the proportion is to use the `mean()` function.

We can also talk about the odds of success, i.e., that the odds of heads versus tails are one to one, or 1:1. The odds of it raining on a given day in Glasgow would be 170:195; the denominator is the number of days it did not rain (365 - 170 = 195). Expressed as a decimal number, the ratio 170/195 is about 0.87, and is known as the **natural odds**. Natural odds ranges from 0 to $+\inf$. Given Y successes on N trials, we can represent the natural odds as $\frac{Y}{N-Y}$. Or, given a probability p, we can represent the odds as $\frac{p}{1-p}$.

The natural log of the odds, or **logit** is scale on which logistic regression is performed. Recall that the logarithm of some value Y gives the exponent that would yield Y for a given base. For instance, the log_2 (log to the base 2) of 16 is 4, because $2^4 = 16$. In logistic regression, the base that is typically used is e (also known as Euler's number). To get the log odds from odds of, say, Glasgow rainfall, we would use `log(170/195)`, which yields -0.1372011; to get natural odds back from log odds, we would use the inverse, `exp(-.137)`, which returns about 0.872.

8.4.2 Properties of log odds

log odds $= \log\left(\frac{p}{1-p}\right)$

Log odds has some nice properties for linear modeling.

First, it is symmetric around zero, and zero log odds corresponds to maximum uncertainty, i.e., a probability of .5. Positive log odds means that success is more likely than failure (Pr(success) $>$ .5), and negative log odds means that failure is more likely than success (Pr(success) $<$ .5). A log odds of 2 means that success is more likely than failure by the same amount that -2 means that failure is more likely than success. The scale is unbounded; it goes from $-\infty$ to $+\infty$.

8.4.3 Link and variance functions

The link function for logistic regression is:

$$\eta = \log\left(\frac{p}{1-p}\right)$$

while the inverse link function is:

$$p = \frac{1}{1+e^{-\eta}}$$

where e is Euler's number. In R, you could type this latter function as `1/(1 + exp(-eta))`.

The variance function is the variance for the binomial distribution, namely:

$$np(1-p)$$

.

The app below allows you to manipulate the intercept and slope of a line in log odds space and to see the projection of the line back into response space. Note the S-shaped ("sigmoidal") shape of the function in the response shape.

```
## PhantomJS not found. You can install it with webshot::install_phantomjs(). If it is
```

Logistic regression web app https://shiny.psy.gla.ac.uk/Dale/logit

8.4.4 Estimating logistic regression models in R

For single-level data, you use the `glm()` function. Note that it is much like the `lm()` function you are already familiar with. The main difference is that you specify a `family` argument for the link/variance functions. For logistic regression, you use `family = binomial(link = "logit")`. The logit link is default for the binomial family with a logit link, so typing `family = binomial` is sufficient.

```
glm(DV ~ IV1 + IV2 + ..., data, family = binomial)
```

For multi-level data where there are random effects to be modeled, you use the `glmer` function from `lme4`:

```
glmer(DV ~ IV1 + IV2 + ... (1 | subject), data, family = binomial)
```

Chapter 9

Modeling Ordinal Data

It is perhaps easiest to think about ordinal data by viewing it as a more general case of logistic regression. In logistic regression, the response variable Y has two categories (e.g., 0, 1), with the model intercept representing the point in log odds space where 0 and 1 are equiprobable.

$$\eta = \beta_0 + \beta_1 X_i$$

```
library(tidyverse)
```

Appendix A

Symbols

A.1 General notes

- Greek letters represent population parameter values; roman letters represent sample values.
- A Greek letter with a "hat" represents and *estimate* of the population value from the sample; i.e., $\mu_x$ represents the true population mean of $X$, while $\hat{\mu}_x$ represents its estimate from the sample.

A.2 Table of symbols

symbol	pronunciation	definition
$\mu$	meeYU	generally, a population mean; sometimes, a model intercept. $\mu_x$ r
$\sigma$	sigma	lower case sigma is the standard deviation; $\sigma_x$ is the standard
$\sigma^2$	sigma squared	variance
$\rho$	row	population correlation; $\rho_{xy}$ is the correlation in the population
$r$	row	sample correlation; r_{xy} is the correlation in the sample between $X$
$\mathbf{\Sigma}$	sigma	the capital letter sigma in boldface represents a variance-covariance mat
$\sum$	sigma	upper case sigma is an instruction to add; e.g., $\sum X_i$ is the instru
$\beta$	beta	regression coefficient
$\sim$	is distributed as	e.g., $e \sim N\left(\mu, \sigma^2\right)$ means that $e$ is distributed
$\gamma$	gamma	fixed effects, correlation coefficients in a mixed-effects regression
$\tau$	tau	by-subject variance component (random effects parameter) in a mixed-ef
$\omega$	omega	by-stimulus variance component (random effects parameter) in a mixed-
$S_{0s}$	S sub zero S	by-subject random intercept effect for subject $s$
$S_{1s}$	S sub one S	by-subject random slope effect for subject $s$

Appendix B

Bibliography

Bibliography

Baayen, R. H., Davidson, D. J., and Bates, D. M. (2008). Mixed-effects modeling with crossed random effects for subjects and items. *Journal of memory and language*, 59(4):390–412.

Barr, D. J. (2013). Random effects structure for testing interactions in linear mixed-effects models. *Frontiers in psychology*, 4:328.

Barr, D. J. (2017). Generalizing over encounters: Statistical and theoretical considerations. In *Oxford Handbook of Psycholinguistics*. Oxford University Press.

Barr, D. J., Levy, R., Scheepers, C., and Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of memory and language*, 68(3):255–278.

Bates, D., Maechler, M., Bolker, B., and Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, 67:1–48.

Belenky, G., Wesensten, N. J., Thorne, D. R., Thomas, M. L., Sing, H. C., Redmond, D. P., Russo, M. B., and Balkin, T. J. (2003). Patterns of performance degradation and restoration during sleep restriction and subsequent recovery: A sleep dose-response study. *Journal of Sleep Research*, 12(1):1–12.

Clark, H. H. (1973). The language-as-fixed-effect fallacy: A critique of language statistics in psychological research. *Journal of verbal learning and verbal behavior*, 12(4):335–359.

Coleman, E. B. (1964). Generalizing to a language population. *Psychological Reports*, 14(1):219–226.

DeBruine, L. and Barr, D. J. (2020). Understanding mixed effects models through data simulation. *Advances in Methods and Practice in Psychological Science*.

Judd, C. M., Westfall, J., and Kenny, D. A. (2012). Treating stimuli as a random factor in social psychology: A new and comprehensive solution to a pervasive but largely ignored problem. *Journal of personality and social psychology*, 103:54–69.

Luke, S. G. (2017). Evaluating significance in linear mixed-effects models in r. *Behavior Research Methods*, 49:1494–1502.

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, H., and Bates, D. (2017). Balancing Type I error and power in linear mixed models. *Journal of Memory and Language*, 94:305–315.

McElreath, R. (2020). *Statistical Rethinking: A Bayesian course with examples in R and STAN*. CRC Press.

Yarkoni, T. (2019). The generalizability crisis.