The brms Book: Applied Bayesian Regression Modelling Using R and Stan (Early Draft)

Paul-Christian Bürkner

2024-10-01

Table of contents

Preface

Welcome to this early draft of my brms book. I continue to add new chapters as they get to a sufficiently complete and readable state. I don't necessarily write chapters in the order they will appear in the final version. So please be aware that chapters may change their number or even their order. If you are looking for any updates on the book' status, please check out https://paulbuerkner.com/software/brms-book.

1 Linear Models

1.1 Setup

```
library(magrittr)
library(posterior)
library(ggplot2)
library(bayesplot)
library(brms)
```

1.2 Introduction

Linear models are the most simple regression models we can come up with. Yet, they are already very powerful but also surprisingly complex. For example, most of the discussion around causal modeling takes place in the linear modeling space and few people would claim that topic to be particularly easy.

In this chapter, will analyze the `epilepsy` data set that contains information about seizure counts in a randomized trial of anti-convulsant therapy in epilepsy. This data was initially provided by Thall and Vail (1990) and later used as an example in Breslow and Clayton (1993).

```
data("epilepsy", package = "brms")
```

count	Trt	Base	Age	patient	visit
5	0	11	31	1	1
3	0	11	30	2	1
2	0	6	25	3	1
4	0	8	36	4	1
7	0	66	22	5	1
5	0	27	29	6	1

Our response variable to be predicted is the number of seizure counts `count` within a given time interval. As predictors we have treatment (variable `Trt`; dummy-coded), 8-week baseline seizure counts (`Base`), and age of the patients in years (`Age`). Additionally, since each of the 59 `patients` is measured four times over the course of the study, we have an additional `visit` variable running from 1 to 4. We will ignore the `patient` and `visit` information for now but revisit it in later chapters.

As you may have already thought by yourself, linear regression is not actually well suited for analyzing the `epilepsy` data. In my view, this coincides with most cases encountered in the real world, where linear regression can be considered only a very crude approximation at best. For a textbook example, this dataset is actually quite inconvenient as we will see. But let's be honest, you are here to learn advanced Bayesian regression modeling. You have already chosen the hard path and probably don't care so much about the convenience of your statistical analysis.

1.3 Gaussian linear models

Below, as well as in the upcoming chapters, we will denote response variables as y, predictor variables (aka covariate or feature) as x, and the observation index as n, which runs from 1 to the total number of observations N. With this notation, we can write the likelihood of a simple (Gaussian) linear model as

$$y_n \sim \text{normal}(\mu_n, \sigma) \tag{1.1}$$

$$\mu_n = b_0 + b_1 x_n. \tag{1.2}$$

In this book, I will write the (univariate) normal distribution in terms of mean μ and standard deviation σ. In the literature, you may sometimes see variance σ^2 or precision $1/\sigma^2$ instead of the standard deviation, so don't be confused if you see other people writing the normal distribution a bit differently. I prefer to write things in terms of the standard deviation because it is on the scale of the variable being normally distributed rather than on a quadratic (or inverse quadratic) scale, whose interpretation I consider harder.

With respect to our linear model, the parameters of are the intercept b_0, the slope b_1 corresponding to x, which together form the linear predictor (mean) μ, as well as the residual standard deviation σ capturing the errors (noise) that cannot be explained by the linear predictor. An equivalent formulation of Equation 1.1 that some of you may be more familiar with is

$$y_n = \mu_n + \varepsilon_n \tag{1.3}$$

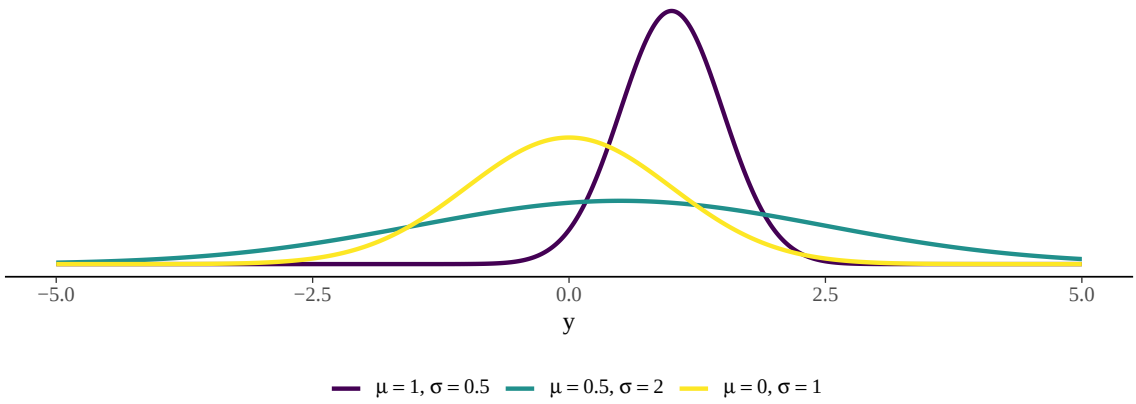


Figure 1.1: Three example densities of the normal distribution varying in both mean μ and standard deviation σ.

$$\varepsilon_n \sim \text{normal}(0, \sigma) \tag{1.4}$$

However, the latter formulation does not generalize well to many non-normal models, so you will need to get comfortable with the former way of writing it.

Since this book is about Bayesian modeling, you may expect a discussion of priors coming up next, but I won't bother you with that just yet. The simple linear model we are considering is way too simple compared to the richness of the data for prior specification to be practically needed. In my experience, by focusing on priors too early, we (as teachers of Bayesian methods) do more harm than good, because doing so tends to shy away lots of users who would actually benefit from Bayesian methods. To be clear, prior specification can be very useful or may even be required in some cases. But for simple examples as the ones presented here, we will be fine without them. Or rather, we can hopefully trust brms to have made reasonable default choices in that regard.

So let's dive right into the brms specification of models. No matter how complex the model you fit, you will always do it through the `brm` function. Its first argument specifies the linear predictor via a `formula`. It can become quite complex but for our simple model here, it just reads `count ~ 1 + Trt`. This is standard R formula syntax. The response variable is on the left-hand side of the `~` and the predictors are on the right-hand side. The symbol `1` explicitly indicates the presence of an intercept, but we could also leave it out without changing the model, since an intercept is included by default anyway. The second argument `data` needs to be a `data.frame` from which the variables are taken. And that's it. We have everything to complete the specification of our first brms model:

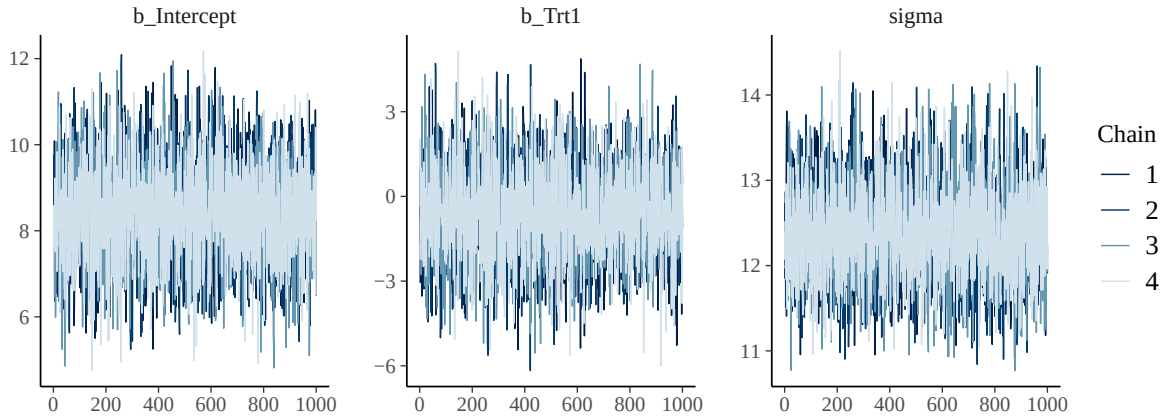
```r
fit_epi_gaussian1 <- brm(count ~ 1 + Trt, data = epilepsy)
```

As you run this code yourself, you will see some output popping up in the process that looks something like this:

```
Compiling Stan program...
Start sampling

SAMPLING FOR MODEL 'XYZ' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 3.7e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition ...
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 1: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 1: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 1: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.023608 seconds (Warm-up)
Chain 1:                0.016017 seconds (Sampling)
Chain 1:                0.039625 seconds (Total)
Chain 1:

<more output from other chains>
```

After model fitting has completed, we can, for example, use trace plots as an initial, graphical convergence check:

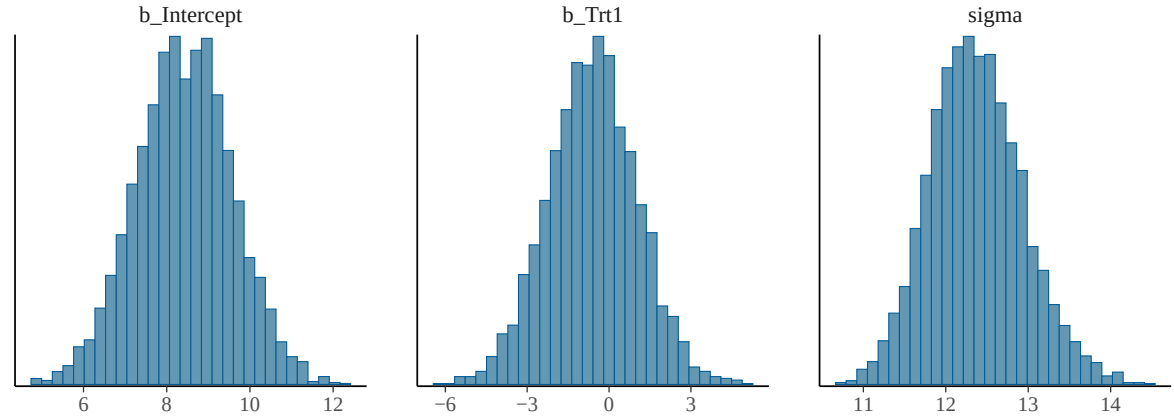
```r
mcmc_plot(fit_epi_gaussian1, type = "trace")
```



In the above plot, we see, for each parameter separately, traces of the four MCMC chains in different colors with post-warmup iterations on the x-axis and parameter values on the y-axis. These trace plots are showing ideal convergence: All chains are overlaying each other nicely, are stationary (horizontal on average), and show little autocorrelation (moving up and doing very quickly).

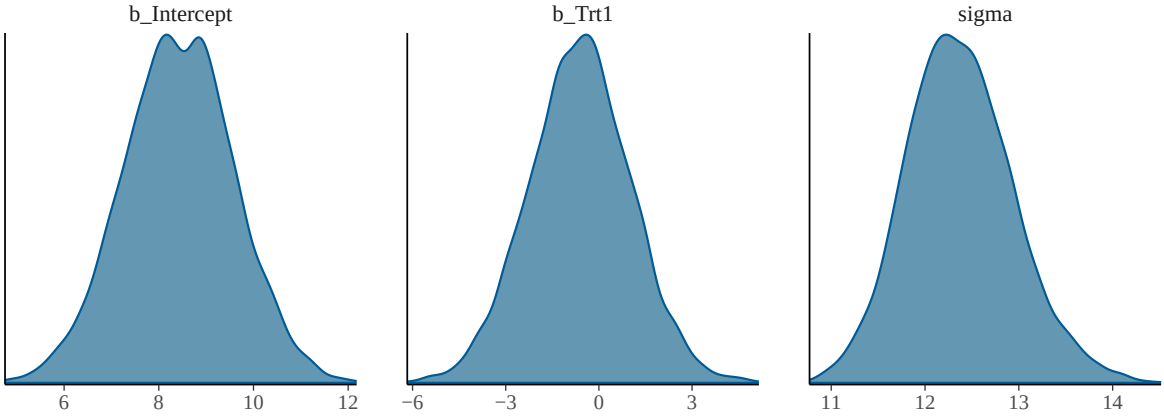
Having convinced us of convergence, at least graphically for now, we can move on to inspecting the posterior. For example, we can plot histograms of the posterior samples per parameter:

```
mcmc_plot(fit_epi_gaussian1, type = "hist", bins = 30)
```



Alternatively, we can apply some density estimation algorithm to give us an approximate density on the basis of the posterior samples:

```
mcmc_plot(fit_epi_gaussian1, type = "dens")
```



These densities should be understood only as a pretty visualization of the estimated posterior, but not be used for actual inference (more on that later).

Visualizations are nice, but we will also want to summarize the posterior samples and their convergence numerically. This is what the `summary` method is for.

```
summary(fit_epi_gaussian1)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: count ~ 1 + Trt
   Data: epilepsy (Number of observations: 236)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     8.42      1.15     6.14    10.68 1.00     3915     2817
Trt1         -0.61      1.63    -3.84     2.59 1.00     4630     3006

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    12.37      0.57    11.32    13.58 1.00     3792     2648

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

There is a lot to unpack here in the summary output. We start with some aspects and come back to others in later chapters.

At the top of the output, we see a short overview over the most important input, we have given to the model, for example the formula and dataset. Moving to the actual posterior results, brms shows two blocks of parameters here, namely `Regression Coefficients` and `Further Distributional Parameters`. The former name is self-explanatory I think. The latter refers to additional parameters of the likelihood that we did not predict with our regression, here the residual standard deviation `sigma`. For each of the displayed parameters (rows), we see seven summary statistics (columns). The `Estimate` column is simply the empirical mean of the posterior draws $\bar{\theta} = \frac{1}{S}\sum_{s=1}^{S}\theta^{(s)}$, which is a sampling-based estimate of the analytical posterior mean. The `Est.Error` columns denotes (the empirical estimate of) the posterior standard deviation, while the lower and upper bounds of the 95% credible interval are computed as the 2.5% and 97.5% quantile of the posterior samples, respectively. The three remaining columns, `Rhat`, `Bulk_ESS`, and `Tail_ESS` give us information about convergence and sampling efficiency, as also shortly explained at the bottom of the summary output. I will go into more details of convergence diagnostics later on in Chapter X. At this point, all you need to know is that convergence is basically perfect for this model with `Rhat` being `1.00` and ESS estimates being roughly as large as the total number of post warmup draws $S = 4000$.

1.3.1 Why do we need sampling?

A common understanding is that we are using sampling-based methods only because we do not have access to the analytic posterior. While this is certainly the primary reason, it turns out that posterior draws are actually way easier to handle than posterior densities as I will illustrate with a simple example. Suppose we are interested in the posterior of the residual variance σ^2 while we only have information on the residual standard deviation σ in our model. Fortunately, when the posterior is stored in the form of draws, this is no problem at all. We just have to apply the desired transformation *for each draw separately* and, voilà, we have posterior draws for our transformed quantity of interest. In our examples, this means that the squared posterior draws of σ are in fact valid posterior draws of σ^2:

$$\left(\sigma^{(s)}\right)^2 = \left(\sigma^2\right)^{(s)}$$

But isn't this obvious you may ask? Well, not necessarily, at least once we consider the effort we would have to put into this transformation if we only had posterior densities instead. The key term here is *Jacobian adjustment*. I am not explaining the details here, but you can look them up, for example, in Gelman et al. (2013) to convince yourself that dealing with densities when transforming parameters is quite a lot of work.

The reason we need a Jacobian adjustment above is that the square is a non-linear transformation. But even if our transformation is linear, having access to draws makes our live considerably easier. Suppose we are interested in the posterior of the expected number of seizures in the treatment group (call it μ_{Trt}), then we don't have this quantity in our model from above. But we have everything we need: Because the treatment variable was contrast

coded, we know that $\mu_{\text{Trt}} = b_0 + b_1$. Accordingly, all we have to do to get posterior draws of μ_{Trt} (and thus a representation of its posterior) is to apply the above transformation *for each draw separately*:

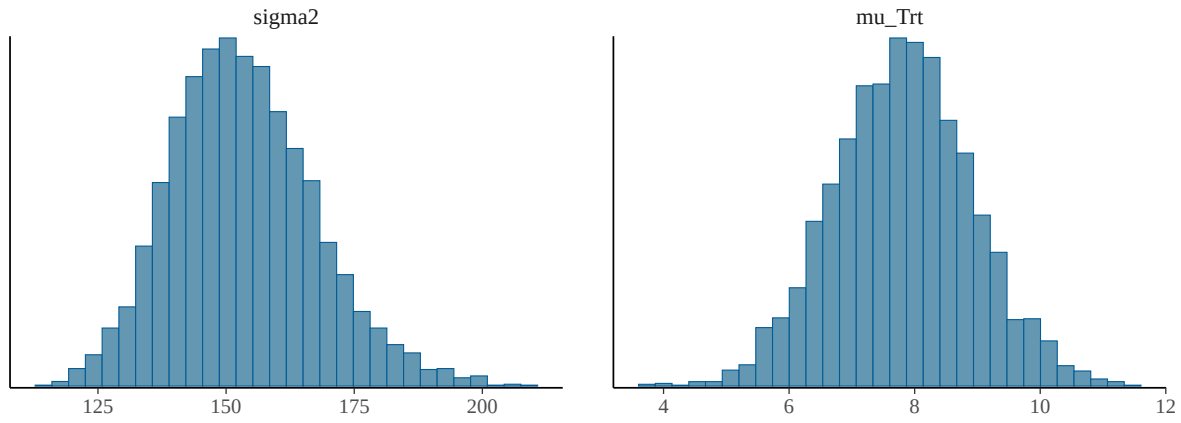
$$\mu_{\text{Trt}}^{(s)} = b_0^{(s)} + b_1^{(s)}$$

In R, we can easily perform such transformations by first extracting the posterior draws and then applying the transformation per draws in a vectorized manner. For example, using the functionality from the **posterior** package:

```
draws <- as_draws_df(fit_epi_gaussian1) %>%
  mutate_variables(sigma2 = sigma^2,
                   mu_Trt = b_Intercept + b_Trt1)
```

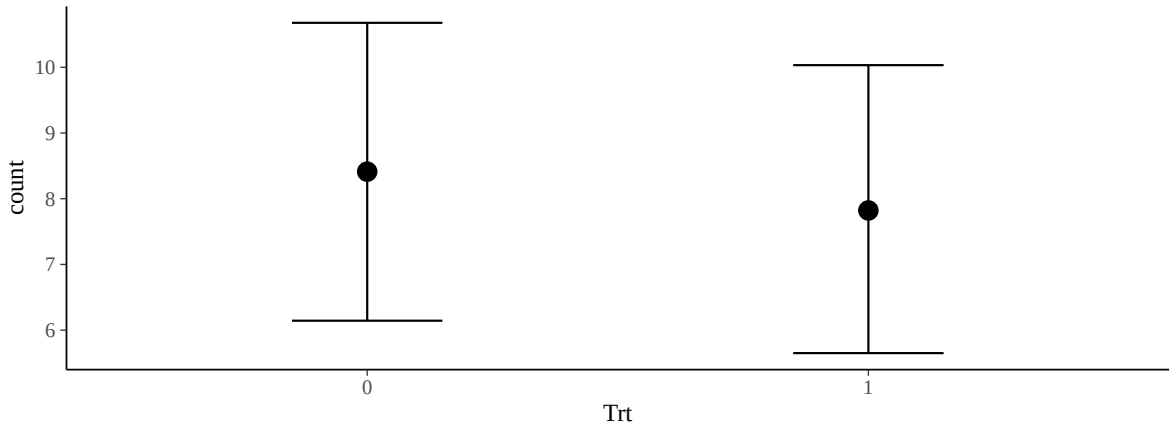
Plotting can then be performed for example via the **bayesplot** package, which is also implicitly used in the `mcmc_plot` function applied earlier.

```
bayesplot::mcmc_hist(draws, c("sigma2", "mu_Trt"), bins = 30)
```



With μ_{Trt} we computed the model-implied predictions of the mean for the treatment group. For the control group, it would just be $\mu_{\text{Ctrl}}. = b_0$ for this simple model. For more complex models, computing the predictions for different groups or, more generally, different predictor values manually becomes quite cumbersome. For this reason **brms** provides you with a convenient method to provide quick graphical summarizes of the model-implied predictions per predictor:

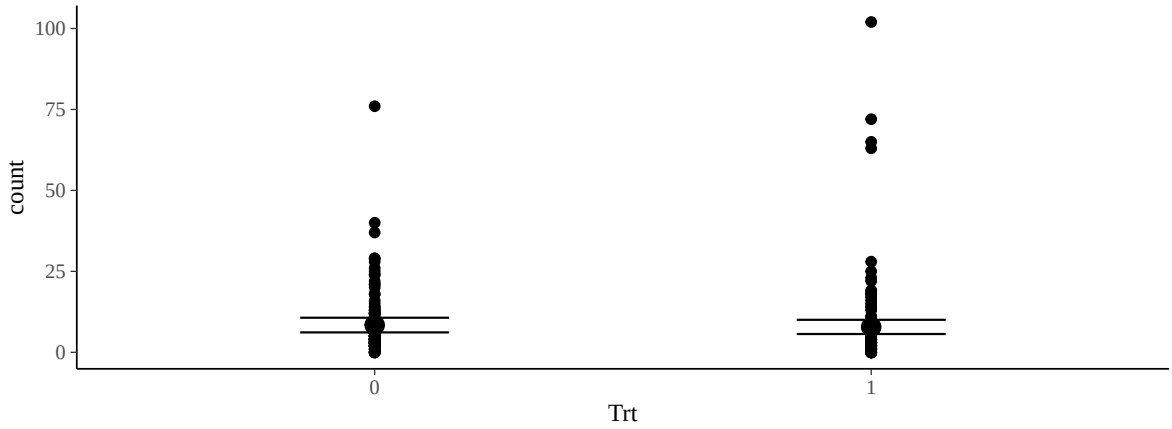
```
ce <- conditional_effects(fit_epi_gaussian1)
plot(ce)
```



The error bars are representing 95% credible intervals by default, but we can change that value if we like via the `prob` argument. Comparing the `conditional_effects` plot with our manually computed posterior of μ_{Trt}, we see that they actually are the same.

To put the mean predictions into the context of the observed data, we can also show the data as points in the plot:

```
plot(ce, points = TRUE)
```



Clearly, there are some points outlying points in both groups, which makes the differences in group means hard to see. Accordingly, depending on our goal, it may or may not make sense to show the data points.

For the remainder of this chapter, we would like to always see the points in `conditional_effects` so we are setting the corresponding option globally to shorten our code.

```r
options(brms.plot_points = TRUE)
```

1.3.2 Variations of posterior predictions

What we do in `conditional_effects` by default, and have done above, is visualizing the *expected value* (mean) of the posterior predictive distribution, conditional on certain predictor values. In brms, this is done via the `posterior_epred` (posterior expected predictions) method. For example, we can run

```r
newdata <- data.frame(Trt = c(0, 1))
pe <- posterior_epred(fit_epi_gaussian1, newdata = newdata)
```

to create expected posterior predictions for both groups. The resulting object contains the posterior draws in the rows and the different conditions (here, treatment groups) in the columns

```r
str(pe)
```

```
 num [1:4000, 1:2] 9.02 7.82 8.02 10.09 6.92 ...
```

We can summarize the draws, for example, via

```r
posterior_summary(pe)
```

Estimate	Est.Error	Q2.5	Q97.5
8.42	1.15	6.14	10.68
7.81	1.10	5.65	10.03

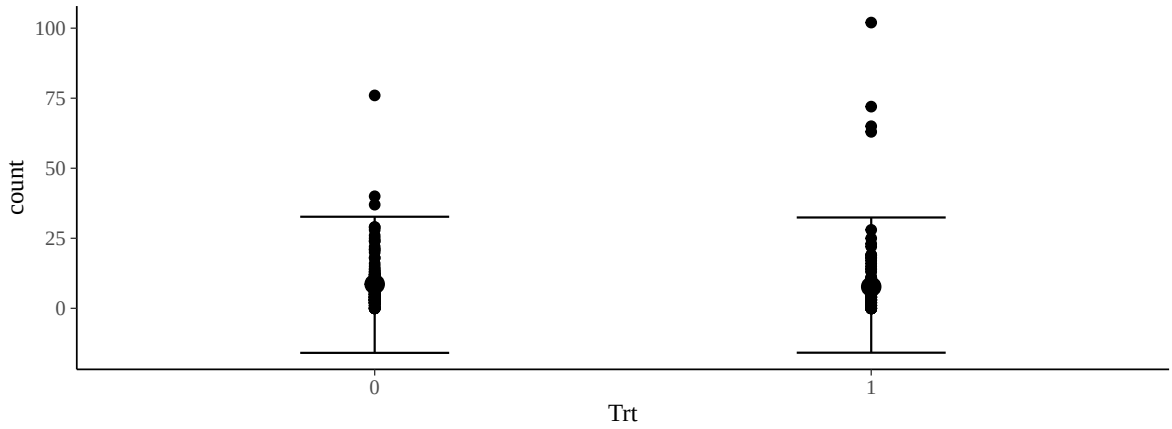
In linear models, `posterior_epred` directly coincides with evaluating the linear predictor μ as exemplified above. What `posterior_epred` does not include is the residual (aleatoric) uncertainty, which is represented by σ in our linear models.

Let's consider again the task of evaluating predictions for the treatment group. If we are only interested in (the posterior of) the expected value of the posterior predictive distribution, we would compute $\mu_{\text{Trt}}^{(s)} = b_0^{(s)} + b_1^{(s)}$. In contrast, if we are interested in prediction of hypothetical new data points $y_{\text{Trt}}^{(s)}$ from the treatment group (i.e., actual posterior predictions), we would sample

$$y_{\text{Trt}}^{(s)} \sim \text{normal}(\mu_{\text{Trt}}^{(s)}, \sigma^{(s)})$$

This is exactly what happens behind the scenes when we execute the code below.

```
conditional_effects(fit_epi_gaussian1, method = "posterior_predict")
```



We could have also done this more manually via

```
newdata <- data.frame(Trt = c(0, 1))
pp <- posterior_predict(fit_epi_gaussian1, newdata = newdata)
```

```
posterior_summary(pe)
```

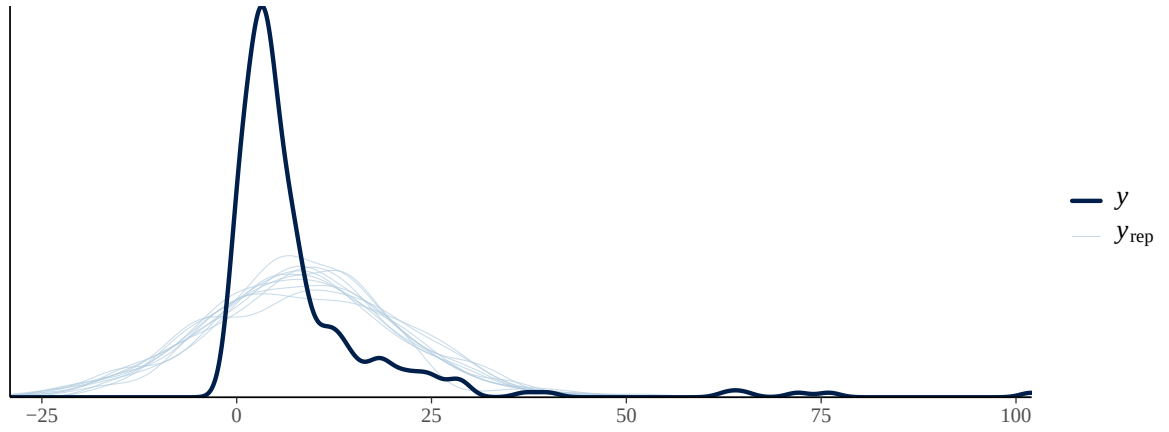
Estimate	Est.Error	Q2.5	Q97.5
8.423633	1.151495	6.143697	10.67606
7.809943	1.103698	5.650650	10.03225

Comparing the visual with the numeric output is a bit difficult here, but be rest assured that both is doing the same thing. One note of caution though: As we sample new $y_{\text{Trt}}^{(s)}$ as part of `posterior_predict`, we will get slightly different results every time we run it, unless we fix R's random seed via the `set.seed` function.

1.3.3 Posterior predictive checks

We are already aware that the considered linear regression model is not ideal for the `epilepsy` data. But how bad is it? As quick graphical method, we can use posterior predictive (PP) checks, where we are compare the observed outcome data with the model predicted outcome data, that is, with the posterior predictions. In brms, we can perform PP-checks via

```
pp_check(fit_epi_gaussian1)
```

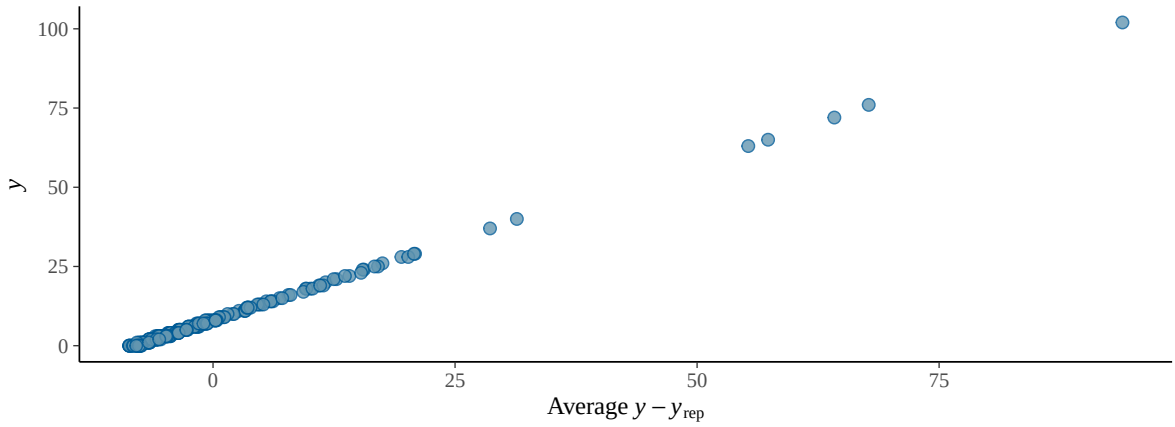


By default, the `dens_overlay` (density overlay) type is used, which plots the marginal density of the observed outcomes (shown in dark color) against marginal densities obtained posterior predictions (shown in lighter color). Each posterior draws implies it's own posterior predictive distribution over observations. Hence, in the above plot, we see ten light blue lines, each corresponding to one posterior draw. We could of course show more such lines, but usually the pattern is clear just from these few.

Back to criticizing our linear model, there is a lot to learn even from this simple PP-check. We see that the model predictions can neither account for the strong spike of observed outcomes close to zero nor for their right-skewness. Instead, the the model also predicts a lot of negative outcomes, which is impossible in reality because we are predicting counts of epileptic seizures. In the plot, it looks as if the observed data also had few negative values (the dark blue density going below zero) but this is just an artifact of estimating a continuous density from counts. While this PP-check type is definitely not ideal to illustrate count outcome data, it still very clearly points to the shortcomings of our linear model.

While the default PP-check was already eye-opening, there are lot of types that can further our understanding of model appropriateness. For example, an often very useful check is obtained by comparing the residuals = observed outcomes - model predictions with the observed outcomes, also known as *residual plot*. In `pp_check` this check type is called `error_scatter_avg`:

```
pp_check(fit_epi_gaussian1, type = "error_scatter_avg")
```



Ideally, as for any residual plot, we want to see a point cloud without a visible relationship between residuals and outcomes. But this is not at all what we see above. Instead, there is a strongly almost perfectly linear relationship indicating strong problems with the independence assumption of the errors. Essentially, both PP-checks have told us that our initial model is a very bad for the data at hand. In fact, we will spend quite a bit of time in this and later chapters to build better models for this data.

Before we move on, one small tip about the `pp_check` method. If you don't know which check types are available, you can simply pass an arbitrary non-supported `type` name to get a list of all currently supported types:

```
pp_check(fit_epi_gaussian1, type = "help_me")
```

1.3.4 Adding more predictors

There are a lot of problems with our initial model. The first problem we will address is the fact that there are other (non-treatment) variables that we want to control for because they have a relevant influence on our outcomes. The treatment was randomized so we need to worry less about confounding than in observational studies, but still it will likely be very beneficial to control for some other variables, even if it was only to reduce the uncertainty in our treatment effects' posterior.

In this example, we will use the number of epileptic seizures observed in a standardized time frame *before* treatment, a variable called `Base`. In `brm` all we have to do is to add it to the right-hand of the model formula:

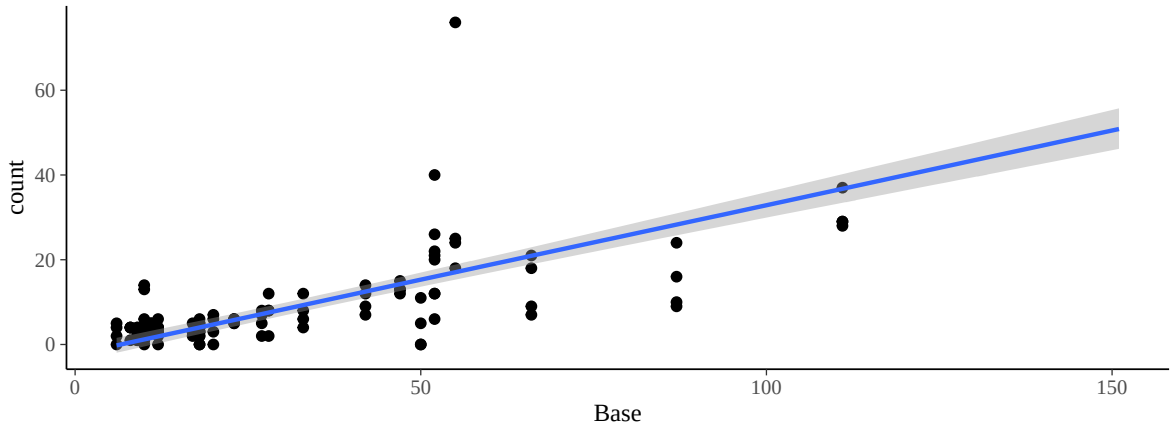
```
fit_epi_gaussian2 <- brm(count ~ Trt + Base, data = epilepsy)
```

```
summary(fit_epi_gaussian2)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: count ~ Trt + Base
   Data: epilepsy (Number of observations: 236)

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept    -2.33      0.96    -4.20    -0.42 1.00     4741     2840
Trt1         -0.90      1.04    -2.91     1.14 1.00     4590     3081
Base          0.35      0.02     0.31     0.39 1.00     4275     3002

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     8.02      0.37     7.33     8.80 1.00     3951     2708
```

The summary output indicates that, unsurprisingly, `Base` is very strongly and positively related to the outcome, which is confirmed by a graphical summary:

```
conditional_effects(fit_epi_gaussian2, effects = "Base")
```



What is more, while the posterior mean of `Trt` has changed only a bit, the posterior uncertainty, for example, as quantified by the 95% credible interval bounds has reduced quite a lot as a result of controlling for `Base`.

The above model assumes that the effect of the treatment is independent of the baseline number of seizures. But what if the treatment worked better (or worse) for patients with

stronger epileptic symptoms, as measured via number of seizures before treatment. This calls for an interaction of `Trt` and `Base`, which we can write in R formula syntax as `Trt * Base`. In fact, as is the case for all standard regression functions in R as well, this will also add the individual coefficients for the two predictors so our interaction model is simply written as:

```
fit_epi_gaussian3 <- brm(count ~ Trt * Base, data = epilepsy)
```

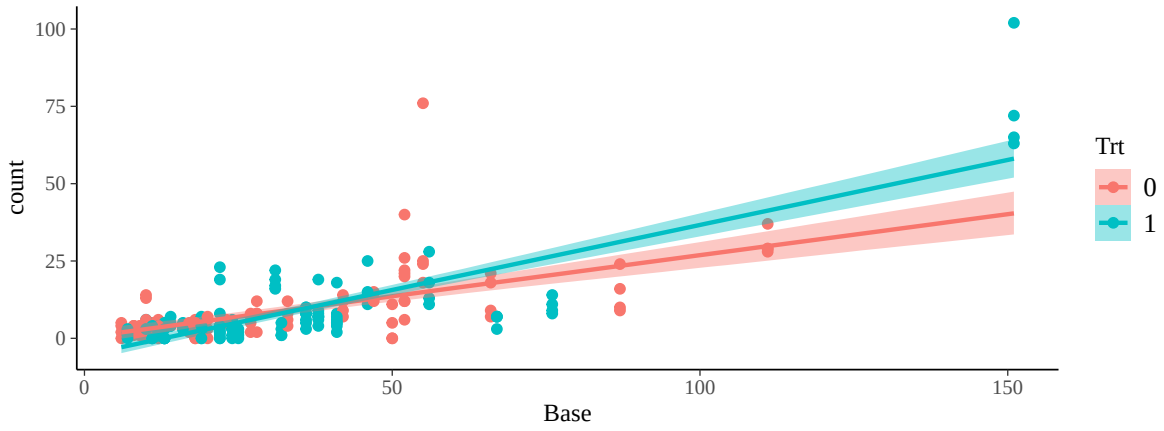
Again, we first summarize the model both numerically:

```
summary(fit_epi_gaussian3)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: count ~ Trt * Base
   Data: epilepsy (Number of observations: 236)

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     0.34      1.16    -1.95     2.68 1.00     2326     2785
Trt1         -5.74      1.59    -8.95    -2.58 1.00     2069     2450
Base          0.27      0.03     0.21     0.32 1.00     2149     2532
Trt1:Base     0.15      0.04     0.08     0.23 1.00     1939     2312

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     7.78      0.36     7.11     8.52 1.00     3528     2423
```

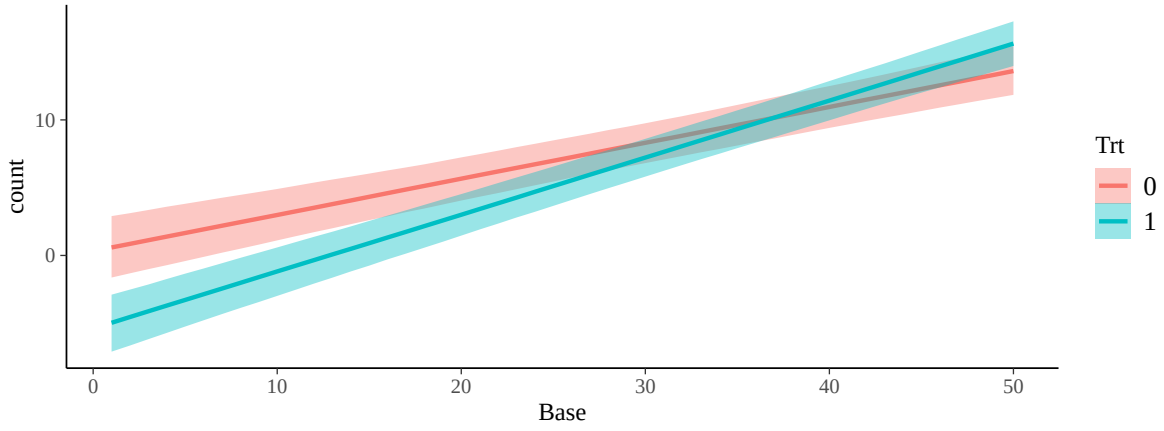
The negative main effect of the treatment points to a beneficial effect for low number of baseline seizures (close to `Base = 0`). However, we also see a positive interaction of treatment and baseline. That is, if we take this result seriously for a moment, the treatment appears to become worse as the baseline seizure count increases. When summarizes graphically

```
conditional_effects(fit_epi_gaussian3, effects = "Base:Trt")
```



we also see that the model predicts the treatment to be worse than the control for very high baseline values. The resolution of the plot is not great to see what happens at lower baseline values. So let's zoom in a bit:

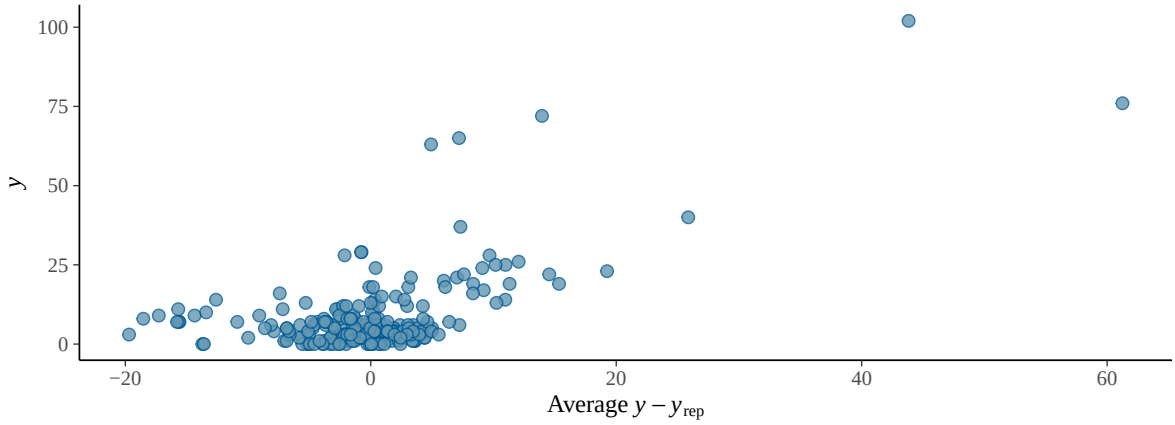
```
conds <- list(Base = seq(1, 50, 1))
ce <- conditional_effects(
  fit_epi_gaussian3, effects = "Base:Trt",
  int_conditions = conds
)
plot(ce, points = FALSE)
```



The argument `int_conditions` (short for interaction conditions) can be used to control which predictor values to plot both on the x-axis and in terms of groups differentiated by color. Now, we more clearly see that the summary has told us, namely that the model expects the treatment to be beneficial as long as the baseline does not exceed around 20 seizures.

Let's take a look at the residual PP-check again:

```
pp_check(fit_epi_gaussian3, type = "error_scatter_avg")
```



We see that things appear less bad as for our initial models, but still not great. In particular, we clearly see some outliers at the top right of the plot, which we could also see in the conditional plots earlier. Knowing that linear models are susceptible to outliers, it is plausible that they have influenced the model in a qualitative manner that affects our conclusions. In the next section, we will investigate this issue in more detail.

1.3.5 Outlier analysis

In Figure 1.2, we see a histogram of the baseline seizure counts we have observed in our data

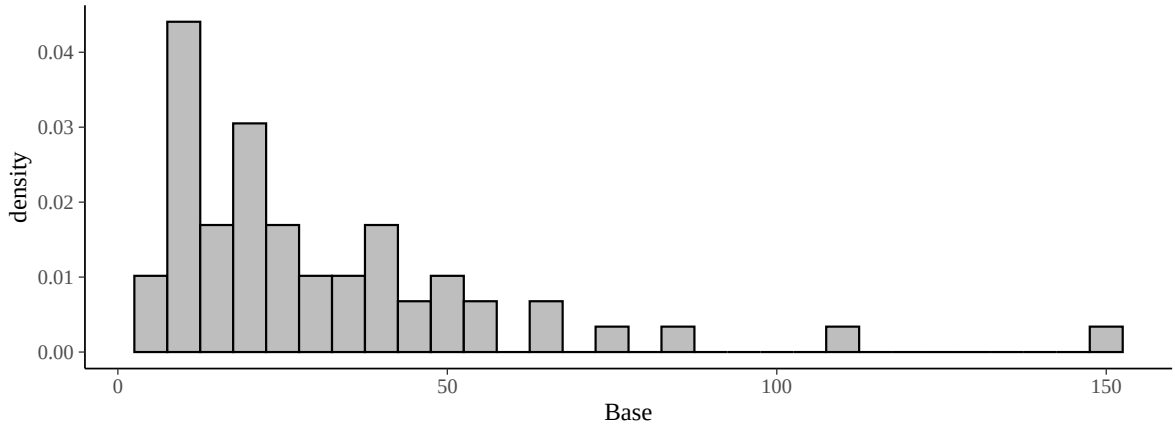


Figure 1.2: Histogram of baseline seizure counts in the epilepsy data set.

There appears to be a group of outliers at `Base = 150` and another one around `Base = 110`. A close investigation reveals that these two groups are belonging to only one person each, whose

baseline values have just been repeated four times due the repeated measures of the outcomes. So removing observation with baseline value above, say, `Base = 140` will in fact only remove a single person from the analysis. Let's see what happens to our model results in this case:

```
fit_epi_gaussian_outlier1 <- brm(
  count ~ Trt * Base,
  data = epilepsy %>% filter(Base < 140)
)
```

When taking a selective look at the regression coefficients we see that the interaction of treatment with baseline has now become negative. That is, after removing data for that one person, the model thinks that the treatment works *better* for people with higher baseline number of seizures, the complete opposite result of what we had before.

	Estimate	Est.Error	Q2.5	Q97.5
b_Intercept	0.38	0.99	-1.55	2.33
b_Trt1	0.42	1.50	-2.58	3.39
b_Base	0.27	0.02	0.22	0.31
b_Trt1:Base	-0.09	0.04	-0.17	-0.01

Let's now also remove data from the second person with outlying baseline scores by filtering out everything above `Base = 100`:

```
fit_epi_gaussian_outlier2 <- brm(
  count ~ Trt * Base,
  data = epilepsy %>% filter(Base < 100)
)
```

The results shown below have changed only a little bit from the first outlier model in that the interaction is now slightly less clearly negative.

	Estimate	Est.Error	Q2.5	Q97.5
b_Intercept	0.51	1.08	-1.58	2.59
b_Trt1	0.33	1.54	-2.71	3.38
b_Base	0.26	0.03	0.20	0.32
b_Trt1:Base	-0.08	0.05	-0.17	0.01

However, if someone followed a strict null-hypothesis significance testing approach, they might actually conclude a qualitative change: Using the 95% credible intervals to declare "statistical

significance" in a frequentist sense, the interaction would have changed from "significantly different from zero" to "not significantly different from zero". I think that declaring scientific discovery by such means is not sensible and I do not advocate it. I just want to point it out in case readers used to such interpretation read this book and wonder about my thoughts on that matter.

Summarizing what we have learned about our epilepsy treatment so far is that it may very well have a positive or negative effect, or something in between, depending on how we look at the data. In other words, using only Gaussian linear models and three variables, we have already managed to utterly confuse ourselves. And we are only getting started modeling this data.

1.4 Fat-tailed linear models

Selectively removing outliers has value in the contexts of sensitivity analyses. But it also increases the number of more or less arbitrary decisions we have to make and thus increases the danger of selective reporting conditional on the outcome of our analyses. What is more, if we have no reason to believe that the outlying data is in some way invalid, then we should model rather than exclude them. So what if we want to continue using linear models for this data without our analysis being too dependent on very few data points?

The flexibility of full Bayesian inference comes to rescue: We can simply replace the normal likelihood with a likelihood that can have fatter tails if the data suggests so. Intuitively, such a likelihood would "expect" outliers to occur, thus not be surprised by their presence and ultimately less influenced by them. A distributions with potentially fatter tails than normal is the *Student-t* distribution (see Figure 1.3). It's density is quite complicated, but we do not have to worry about it, since Stan has a stable and efficient implementation that we can use.

What we do have to care about is the distribution's properties, most notably related to the *degrees of freedom* parameter ν. When ν is small, the Student-t distribution has very fat tails. For $\nu = 1$, the tails are so fat that we cannot even compute the mean of the distribution sensibly anymore (mathematically, we say "the mean does not exist"). The $\nu = 1$ even has a special name: It's called the *Cauchy* distribution. As ν becomes larger, the tails become lighter. Mathematically, Student-t converges to normal as $\nu \to \infty$ but already for $\nu \geq 30$, we can barely see any difference between the two anymore. That is, for practical modeling purposes, it does not matter if $\nu = 30, 100, 10000$ since all of them behave almost like the normal distribution. Accordingly, we care mostly about the space between $\nu = 1$ and $\nu = 30$.

This implies that, in contrast to the regression coefficients b or the scale σ, setting an informative prior on ν is crucial. In particular, we need to avoid sampling issues if the data suggests residuals close to normal. That is, if we didn't specify an informative prior, samples of ν might diverge towards infinity as the Student-t likelihood looks almost the same for any $\nu \geq 30$. Following Juárez and Steel (2010), we set the following prior on ν by default:

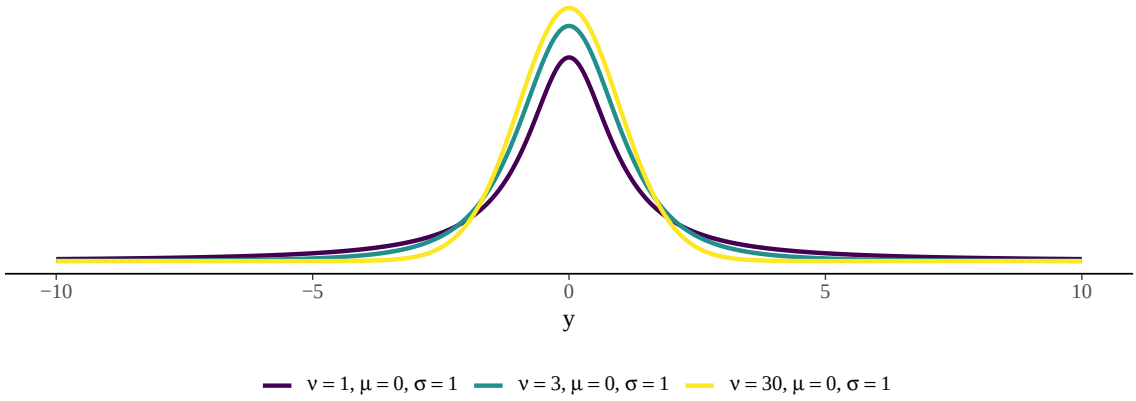


Figure 1.3: Three example densities of the Student-t distribution varying in the degrees of freedom ν.

$$\nu \sim \text{Gamma}(2, 0.1).$$

As displayed in Figure 1.4, most of its mass is assigned to values smaller than 50, thus preventing the posterior to move into problematically high areas ν.

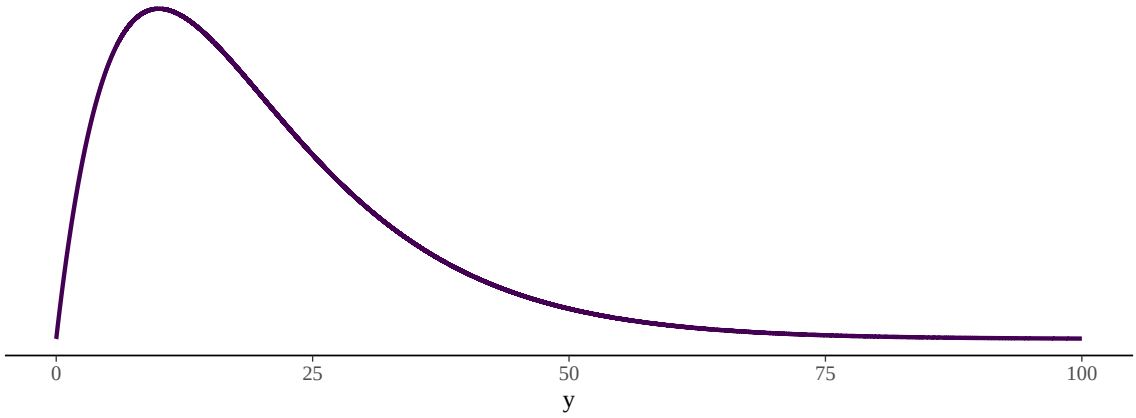


Figure 1.4: Density of the Gamma(2, 0.1) distribution. It is used by brms as the default prior for the degrees of freedom parameter ν of the Student-t likelihood.

To use the Student-t likelihood in brms, all we have to do is to add `family = student()` to the `brm` call. The `family` arguments controls which distribution to use as likelihood. By default `family` is set to `gaussian` so we didn't have to specify it before in our previous models.

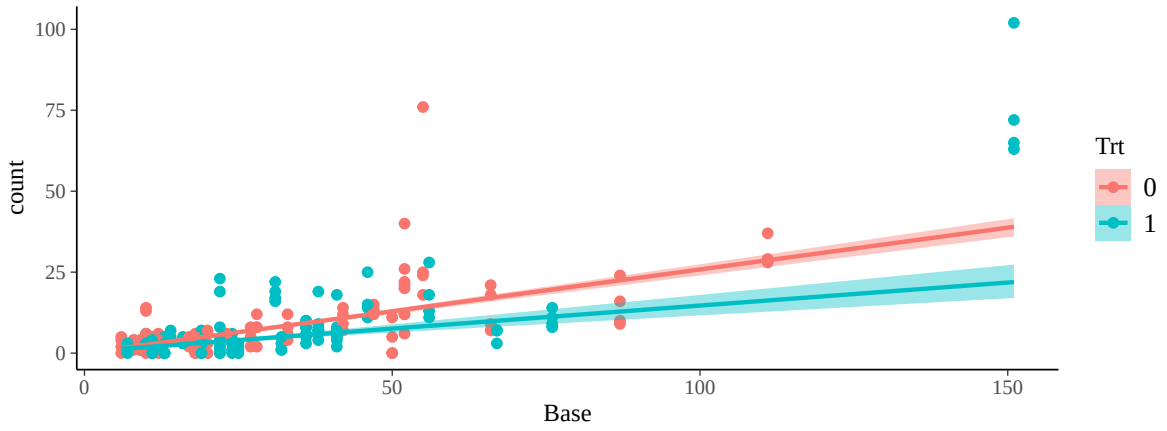
```r
fit_epi_student1 <- brm(
  count ~ Trt * Base,
  data = epilepsy,
  family = student()
)
```

```r
summary(fit_epi_student1)
```

```
 Family: student
  Links: mu = identity; sigma = identity; nu = identity
Formula: count ~ Trt * Base
   Data: epilepsy (Number of observations: 236)

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     0.00      0.36    -0.72     0.70 1.00     3358     3349
Trt1          0.43      0.61    -0.80     1.62 1.00     2072     2494
Base          0.26      0.01     0.24     0.28 1.00     2924     2366
Trt1:Base    -0.12      0.02    -0.16    -0.07 1.00     1971     2300

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     2.10      0.21     1.72     2.54 1.00     2512     2705
nu        1.39      0.18     1.07     1.78 1.00     2075     1185
```

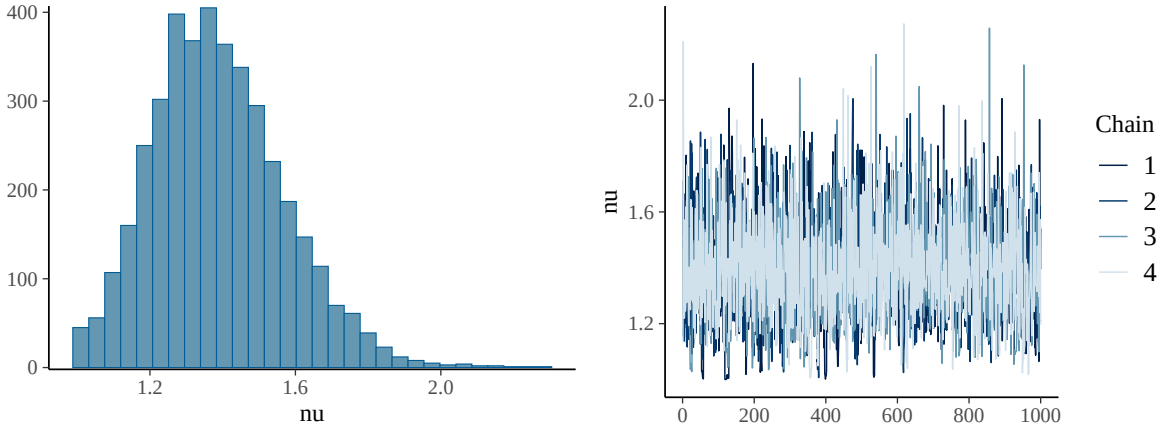
In this summary, two major things stand out. First, both the average effect of treatment and its interaction with the baseline seizure count are clearly negative. This means that the model thinks this treatment is both effective on average and particularly so for higher number of baseline seizures. We can also clearly see this pattern visually:

```r
conditional_effects(fit_epi_student1, effects = "Base:Trt")
```



Second, the summary output reveals that the degrees of freedom parameter `nu` is remarkably small, almost at its lower boundary of 1:

```
plot(fit_epi_student1, variable = "nu")
```



This indicates that, in order for this model to fit the data well, its likelihood had to have very fat tails. This is unsurprising given how extreme some of the outliers are. But it also shows how bad of a likelihood the Gaussian distribution is for this data because it has no statistical mechanism to properly account for these outliers.

We also see in the summary that the scale parameter σ is much smaller in comparison to that we saw in the corresponding normal model ($\sigma \approx 2$ vs. $\sigma \approx 7.8$). This comparison is misleading though. The reason is that, for the Student-t model, σ is no longer directly equal to the residual standard deviation. Instead, we have $\text{SD} = \sigma\sqrt{\frac{\nu}{\nu-2}}$ for $\nu > 2$ and $\text{SD} = \infty$ for $\nu < 2$. Since we have a lot of draws with $\nu^{(s)} < 2$ in the present case, this means that the residual standard deviation estimated by the Student-t model is estimated as infinite or at least very

very large. On a practical level, this is not a realistic estimate for the given data, but the result of the Student-t model's attempt to fit well despite substantial model-misspecification.

1.4.1 Posterior predictive checks

When we run posterior predictive checks with the default `dens_overlay` type, we don't really get something useful at first sight (see Figure 1.5).

```
pp_check(fit_epi_student1)
```

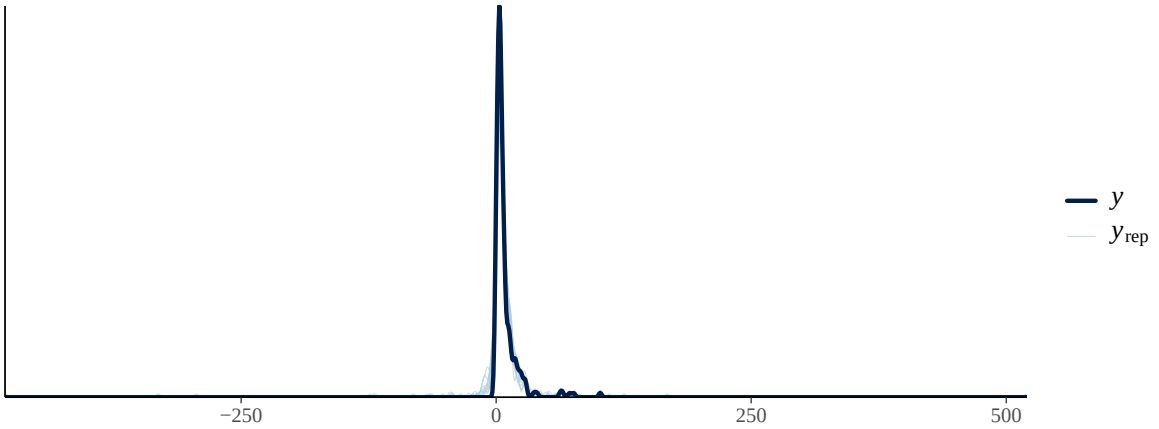


Figure 1.5: Posterior predictive checks of the Student-t model.

This is an unfortunate side-effect of our likelihood being estimated as so fat-failed (small ν). When we generate new data from posterior predictive distribution, some of this data will be sampled as very far away from the bulk of the distribution. While this is to be expected, it effectively destroys the scale of the x-axis of our plot. So let's redo the plot this time truncating the x-axis (see Figure 1.5). Since the `pp_check` method returns ggplot objects, amending the figure is simple:

```
pp_check(fit_epi_student1) + xlim(-50, 150)
```

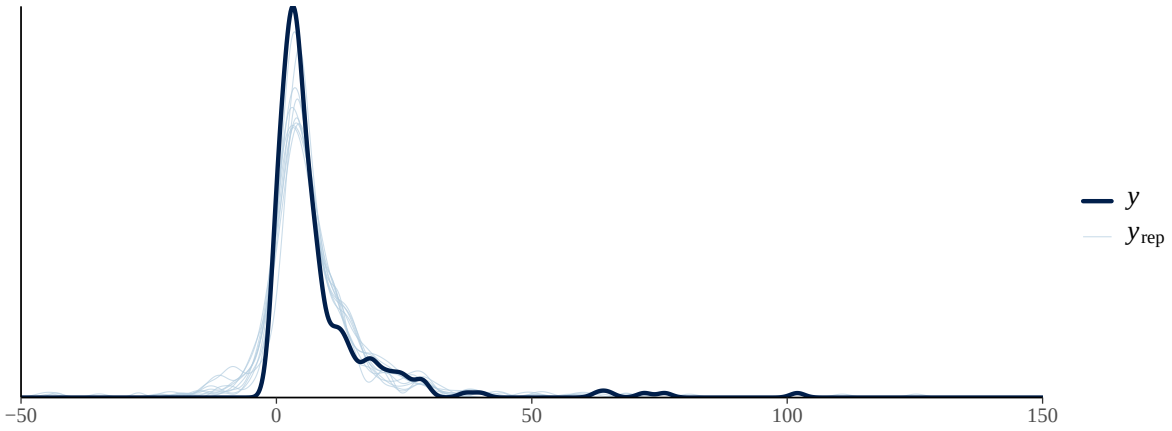


Figure 1.6: Posterior predictive checks of the Student-t model with a truncated x-axis.

Looks much better. And we have only excluded very few predictive draws in the process, so the plot as we see it is still valid. It reveals a surprisingly good marginal in-sample fit of the Student-t model, which can apparently even account for most of the right-skewness of the data distribution. This is only indirectly the related to the use of a Student-t likelihood though, because the likelihood itself remains symmetric. Rather, it is a combination of our linear predictor including the highly right-skewed (and strongly predictive) baseline seizure count in combination with a residual scale σ that is now allowed to be small since the outliers are taken into account separately via ν. The good marginal in-sample fit doesn't necessarily mean our Student-t model is particularly good either. It is apparently just better the normal models we have tried before. We will dive deeper into the topic of model comparison in Chapter 2.

1.5 Skewed linear models

Above, we did come up with the Student-t model to better handle outliers and, as an unexpected side effect, it also turned out to be helping with the skewness. But what if we address the skewness directly via a skewed likelihood family? Here, we will use the skew-normal distribution. Similar to Student-t it generalizes the normal distribution by adding a third distributional parameter only that, for the skew-normal, the third parameter controls the skewness instead of the tail-heaviness.

If you check out text books for details on the skew normal distribution, or Wikipedia, you will usually find it being parameterized in terms of location ζ, scale ω, and skewness α. If α is negative the distribution is left skewed, if α is positive the distribution is right skewed, and if $\alpha = 0$ the skew normal is just equal to the normal distribution.

For non-zero α, location ζ and scale ω are related to the mean and standard deviation only indirectly. Specifically, we can compute the mean μ and the standard deviation σ as

$$\mu = \xi + \omega\sqrt{\frac{2\alpha^2}{\pi(1+\alpha^2)}}, \qquad \sigma = \omega\sqrt{\left(1 - \frac{2\alpha^2}{\pi(1+\alpha^2)}\right)}.$$

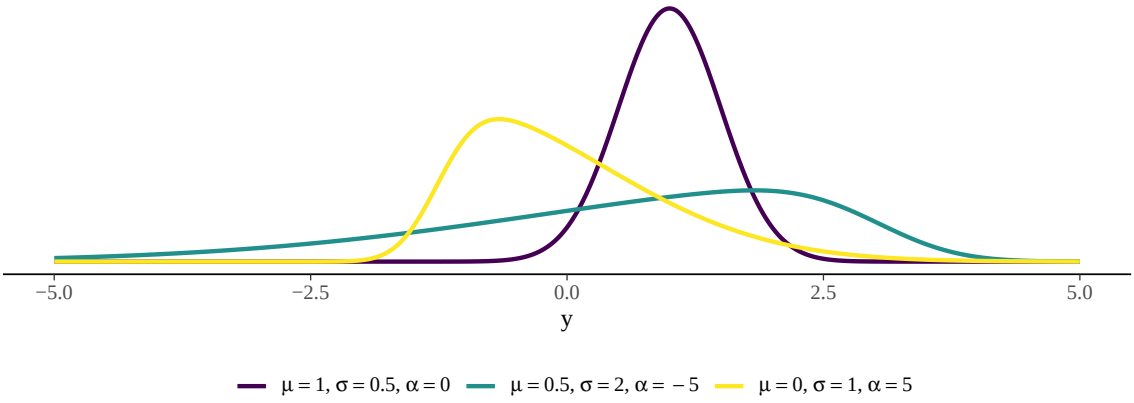


Figure 1.7: Three example densities of the skew-normal distribution varying in mean μ, standard deviation σ, and skewness α.

Especially the fact that $\xi \neq \mu$ (for $\alpha \neq 0$ and $\omega > 0$) makes regression modeling based on this parameterization a bit awkward. We would very much prefer to predict μ because we understand it much better than a scale and skewness affected location. For this reason, `brms` reparameterizes the skew normal in terms of μ, σ, and α, some example densities of which you can see in Figure 1.7. From this mean-standard deviation parameterization, we can recover ξ and ω as follows:

$$\xi = \mu - \omega\sqrt{\frac{2\alpha^2}{\pi(1+\alpha^2)}}, \qquad \omega = \sigma \Big/ \sqrt{\left(1 - \frac{2\alpha^2}{\pi(1+\alpha^2)}\right)}.$$

As a user of `brms`, you don't have to worry about these details. All you need to do is to set `family = skew_normal()`, which ensures that μ, the main parameter to be predicted, is the mean of the likelihood and σ is the corresponding residual standard deviation. For the purpose of analysing the epilpsy data, our skew normal models looks as follows:

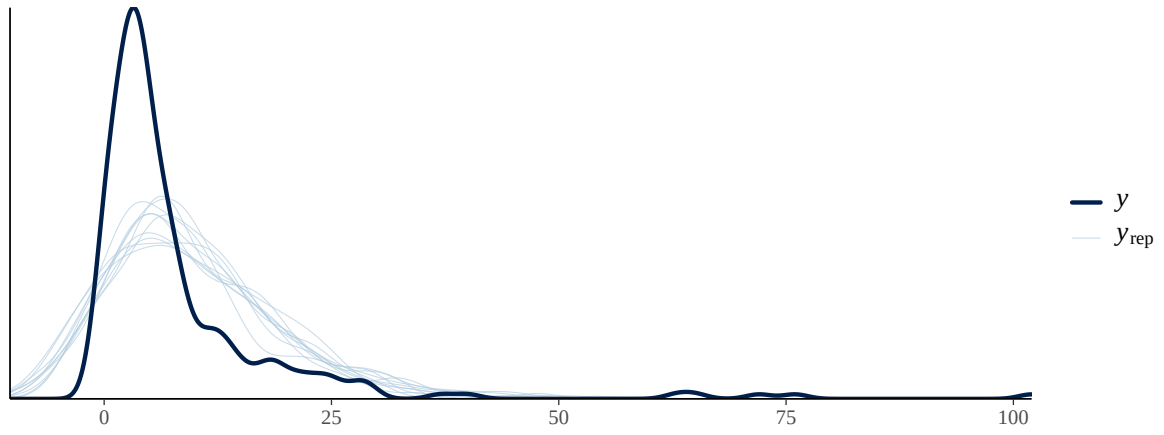
```
fit_epi_skew_normal1 <- brm(
  count ~ Trt * Base,
  data = epilepsy,
  family = skew_normal()
)
```

```
summary(fit_epi_skew_normal1)
```

```
 Family: skew_normal
  Links: mu = identity; sigma = identity; alpha = identity
Formula: count ~ Trt * Base
   Data: epilepsy (Number of observations: 236)

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     5.32      1.12     3.00     7.40 1.00     2214     2209
Trt1         -3.29      1.36    -5.99    -0.63 1.00     2252     2424
Base          0.15      0.03     0.10     0.21 1.00     2244     2672
Trt1:Base     0.07      0.03     0.01     0.13 1.00     2097     2087

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     7.37      0.36     6.70     8.13 1.00     2616     2616
alpha     5.29      1.17     3.40     7.89 1.00     2375     2184
```

From the summary output, we see (a) that the skewness `alpha` is clearly positive, indicating right skewness in the residuals and (b) the interaction of treatment and baseline being positive again. The latter points to the fact that, apparently, the skew normal model is still strongly influenced by the outliers, similar to the normal models, despite trying to account for the right skewness in the data. Let's look at the posterior predictive distribution:

```
pp_check(fit_epi_skew_normal1)
```



From this plot, we clearly see the skew normal model struggles to model the outliers satisfactorily. It clearly predicts right-skewed data, but to a degree that is insufficient for the data at

hand. Having seen this, it is not so surprising anymore that the skew normal model's posterior is much more similar to the normal model's than to the Student-t model's posterior. Within the set of models considered above, we would probably trust the Student-t model most, but keep in mind that none of these models is particularly good yet. We will come back to this later to now focus on more theoretical aspect of Bayesian linear models.

1.6 Gaussian linear models: Analytic vs. MCMC

So far in this chapter, we have looked at the models' posteriors from the approximating lens of MCMC sampling. However, in turns out that Bayesian linear regression is one of the few cases were we can actually compute the posterior analytically. Below, we will us this fact to validate our MCMC approximation. To make the example simpler, we will fix $\sigma = 12$, which is a reasonable value given the data as we have seen above. Then, the only goal becomes to obtain the posterior of the regression coefficients.

For the analytic case, we need a bit of math but I will try to keep it minimal since the purpose is to check the accuracy of the MCMC approximation not the computation of analytic posteriors. We continue to predict the response variable y based on a set of predictors $x_1, x_2, ...$, which we gather as columns in the *design matrix* X. The design matrix also includes an "intercept predictor" $x_0 = (1, 1, ...)$ as first column so that the intercept b_0 can be written as part of $b = (b_0, b_1, ...)$. With this notation, we can write the (vectorized) likelihood of linear regression as

$$y \sim \text{normal}(X\, b, \sigma).$$

For the posterior of linear regression to be analytic, our prior on the coefficient vector b needs to be (multivariate) normal as well. Here, we use a multivariate normal prior with mean μ_0 and covariance matrix $\sigma^2\Sigma_0$:

$$b \sim \text{multi\_normal}(b \mid \mu_{\text{prior}}, \sigma^2\Sigma_{\text{prior}})$$

The multiplication of Σ_{prior} with the residual variance σ^2 is not strictly necessary, but it simplifies the analytic expression of the implied posterior. Perhaps confusingly at first glance, but consistent with common statistics notation, I use mean-standard deviation parameterization for the univariate normal distribution but mean-(co)variance parameterization for the multivariate normal distribution. I will dive deeper into in the context of multilevel models in Chapter 4.

After doing the math to combine likelihood and prior (Gelman et al. 2013), we can see that the analytic posterior of Bayesian linear regression is

$$b \mid y \sim \text{multi\_normal}(b \mid \mu_{\text{post}}, \sigma^2 \Sigma_{\text{post}})$$

with

$$\Sigma_{\text{post}} = (X^T X + \Sigma_{\text{prior}}^{-1})^{-1} \qquad \mu_{\text{post}} = \Sigma_{\text{post}}(\Sigma_{\text{prior}}^{-1} \mu_{\text{prior}} + X^T X \hat{b}),$$

where $\hat{b}$ is the maximum likelihood estimate computed as

$$\hat{b} = (X^T X)^{-1} X^T y.$$

Here, -1 implies the matrix inverse and T the matrix transpose (i.e., switching rows and columns). The formula for μ_{post} illustrates the close relationship of maximum likelihood and fully Bayesian estimation: In linear regression, the posterior mean is essentially a weighted average of the maximum likelihood estimate and the prior mean.

Let's apply these formulas concretely to the model `count ~ 1 + Trt + Base` on the epilepsy data. As priors on the coefficients, we choose

$$b_0 \sim \text{normal}(0, 3), \quad b_1 \sim \text{normal}(0, 5), \quad b_2 \sim \text{normal}(0, 1).$$

That is, before seeing the data, we assume all coefficients to be roughly around zero with varying amounts of (un)certainty. For the intercept, this reflects the assumption that without treatment and with zero previous epileptic seizures, the number of new seizures should be roughly zero. Of course, our prior also allows this number to be negative, something that is impossible in reality. But this is primarily due to the inappropriateness of the normal likelihood for this response variable and cannot (easily) be fixed by the prior distribution. Accordingly, we have to accept for now that our prior implies partially impossible responses. For b_1 and b_2 the prior mean of zero contains some kind of "conservative" assumption that we first want to be convinced by the data before believing that the corresponding predictor is actually relevant. The corresponding standard deviations (5 and 1) reflect our understanding of the predictors' scales. For example, it is unlikely, that the treatment decreases or increases the number of seizures by more than 10 (i.e., twice the standard deviation) on average and it is unlikely that with each baseline seizure, we will see an increase (or decrease) of than 2 seizures during the study. These are all rough considerations though. Prior specification is hard and continues to be a highly active area of research. We will dive deeper into the details of prior specification in later chapters.

In terms of the multivariate normal notation above, these prior choices imply $\mu_0 = (0, 0, 0)$ and $\Sigma_0 = (3, 5, 1)^2/\sigma^2 = (9, 25, 1)/144$ since we decided to fix $\sigma = 12$. With this choice, and after rounding to three digits, the posterior mean and covariance evaluate to

$$\mu_{\text{post}} = (-1.880, -1.071, 0.347) \qquad \sigma^2\Sigma_{\text{post}} = \begin{pmatrix} 1.664 & -0.938 & -0.022 \\ -0.938 & 2.109 & -0.003 \\ -0.022 & -0.003 & 0.001 \end{pmatrix}$$

In terms of the *marginal posteriors* of the coefficients (now rounded to two digits), this implies

$$b_0 \sim \text{normal}(-1.88, 1.29), \quad b_1 \sim \text{normal}(-1.07, 1.45), \quad b_2 \sim \text{normal}(0.35, 0.03).$$

1.6.1 Reproducing the analytic results with MCMC

MCMC is an approximate procedure in the sense that MCMC estimates are based on (dependent) random draws from the posterior and will thus vary slightly depending on which draws we have obtained exactly. How much the estimates are expected to vary is quantified by the *MCMC error*, that is, the (estimated) standard deviation of the distribution of MCMC estimates around their true value. For each parameter, we don't just have a single MCMC error but as many as we have summary statistics. That is, for each parameter, we have a separate MCMC error for the posterior mean, the posterior standard deviation, as well as for every posterior quantile. Independently of whether or not we can obtain the analytic posterior, the MCMC error is a highly useful tool to understand the variation in posterior estimates attributable purely due to algorithmic ("random") fluctuations. The computational details of MCMC errors are quite complex (Vehtari et al. 2021) but for our purposes, the above intuition should be sufficient.

Okay, let's try to reproduce the analytic posteriors with MCMC in brms. For the first time, we will have to specify priors for our model explicitly. An immensely helpful function for this purpose is `get_prior` which you can feed with the regular model input and it will give you all the information you need in order to specify your own priors:

```
get_prior(count ~ 1 + Trt + Base, data = epilepsy)
```

prior	class	coef	group	resp	dpar	nlpar	lb	ub	source
(hidden)	b								default
(hidden)	b	Base							default
(hidden)	b	Trt1							default
(hidden)	Intercept								default
(hidden)	sigma						0		default

The output is a data frame with one prior specification option per row. In the first column, we can find the default priors that brms uses but I have hidden them here since there are not

in focus right now. In the second column, we see the parameter `class` where, for example, `b` stands for regression coefficients. We also see that the intercept b_0 seems to have it's own class (`Intercept`) for reasons that will become apparent soon. In the third column (`coef`), we see the specific coefficient name if we want to set specific priors on specific coefficients. The data frame has a bunch of other columns but we will only make use of them in later chapters. For now, the first few columns are all we need to specify our priors from above:

```
prior_epi_guassian5 <-
  prior(normal(0, 3), class = "Intercept") +
  prior(normal(0, 5), class = "b", coef = "Trt1") +
  prior(normal(0, 1), class = "b", coef = "Base") +
  prior(constant(12), class = "sigma")
```

We can now pass this object to the `prior` argument for `brm` and run the model:

```
fit_epi_gaussian5 <- brm(count ~ 1 + Trt + Base, data = epilepsy,
                         prior = prior_epi_guassian5)
```

To investigate whether the MCMC results match their analytic counterparts up to MCMC error, we need to see both the posterior summaries (here mean and SD) and their corresponding MCMC error estimates. These can be obtained with the help of functions from the **posterior** package:

```
draws <- as_draws(fit_epi_gaussian5, variable = "^b_", regex = TRUE)
summarise_draws(draws, "mean", "sd", "mcse_mean", "mcse_sd")
```

variable	mean	sd	mcse_mean	mcse_sd
b_Intercept	-2.86	1.43	0.02	0.02
b_Trt1	-0.81	1.50	0.02	0.02
b_Base	0.35	0.03	0.00	0.00

When comparing the results, we quickly see that something is not quite right, in particular with the posterior of b_0: Our MCMC estimate of the posterior mean together with its MCMC error are -2.857 $\pm$ 0.023 but the analytic posterior mean is -1.88. This difference is way to large than could be explained by the very small MCMC error. So what has gone wrong?

The answer lies in how brms parameterizes the intercept b_0. By default, the intercept parameter on which we specified the prior above is understood as the intercept when all predictors are centered, that is, have a mean of zero. This is done mainly for reasons of computational efficiency and automatically corrected after model fitting so that the model output contains

the regular intercept as if the predictors had not been centered. Accordingly, as users, we don't have to worry about it unless we want to specify priors on the intercept.

Arguably, specifying priors on the "centered" intercepts is actually simpler because it is immediately related to the mean of the response variable, at least in linear models. In contrast, before centering the predictors, the intercept is often non-sensible, since predictor values of zero may be far away from any actually observed data. For example, if we used participants' age as predictor in a study with adults, an age of zero would not be a sensible value and we would struggle specifying a prior on the resulting intercept when an hypothetical adult had just been born.

Anyway, for the purpose of reproducing the analytical results, we need the standard intercept parameterization without automatic predictor centering. There are multiple equivalent ways to achive this. First, we can use the protected predictor name `Intercept` in the model formula as follows:

```
count ~ 0 + Intercept + Trt + Base
```

Second, we can wrap our regular formula inside the `bf` function (short for `brmsformula`) and explicitly disable the auto-centering:

```
bf(count ~ 1 + Trt + Base, center = FALSE)
```

In both cases, the intercept will now behave like any other regression coefficient also when it comes to prior specification, so we can specify priors on it via `class = "b"` and `coef = "Intercept"`:

```
prior_epi_guassian6 <-
  prior(normal(0, 3), class = "b", coef = "Intercept") +
  prior(normal(0, 5), class = "b", coef = "Trt1") +
  prior(normal(0, 1), class = "b", coef = "Base") +
  prior(constant(12), class = "sigma")
```

```
fit_epi_gaussian6 <- brm(bf(count ~ 1 + Trt + Base, center = FALSE),
                         data = epilepsy, prior = prior_epi_guassian6)
```

Again, we extract the posterior summaries together with their MCMC errors:

```
draws <- as_draws(fit_epi_gaussian6, variable = "^b_", regex = TRUE)
summarise_draws(draws, "mean", "sd", "mcse_mean", "mcse_sd")
```

variable	mean	sd	mcse_mean	mcse_sd
b_Intercept	-1.90	1.28	0.03	0.02
b_Trt1	-1.06	1.41	0.03	0.02
b_Base	0.35	0.03	0.00	0.00

These MCMC estimates are now very close to the analytic posterior summaries for all three parameters. The small deviations are well within a range of $\pm 2\times$ the corresponding MCMC error. In other words, at least for this model and dataset, we have validated our MCMC-based approximation against the analytic posterior. However, this validation only works for a few model classes, where the likelihood and prior match in just the right way for the posterior to have a closed form. For all other cases, we need entirely different procedures for MCMC validation, something we will discuss in more detail in **?@sec-bayesian-workflow**.

1.7 Summary

In this chapter, we have learned about the basics of Bayesian linear models. In the process, we have seen a lot of general principles and methods of Bayesian modeling and how to realize them in brms. This includes not only the core model specifications, but also the principles of sampling, posterior predictions, and some variations of likelihoods and priors. All of these principles and methods will continue to be relevant across this book.

2 Bayesian Model Comparison

2.1 Setup

2.2 Introduction

In practice, we will almost never fit only a single model to our data. Too large is the space of potential assumptions we could make, or choose not to make, and too big are the a priori unknowns such that just knowing "the best model" right away is implausible[1]. Thus, we need to be able to somehow compare multiple models which each other in order to come up with a model (or a set of models) whose inference we choose to trust.

Model comparison is not an easy topic. Despite– or perhaps because of – having worked quite a bit on this topic, I am not feeling super comfortable writing about it. There are just so many things to keep in mind and take into account when comparing, choosing, or averaging over models. Many of you may take my thoughts expressed here seriously and follow my suggestions in their work. Just let me say that often, your understanding of the science, study design, and data will be as valuable in coming up with reasonable models as the statistical comparison methods, which tend to be a bit blind to the specifics of your case.

2.3 Prediction vs. Explanation

Before starting to compare models, we should first remind us about our actual modeling goals. Often we are so stuck in the customs and traditions in our field that we forget to consciously map our research questions to modeling goals and to subsequently build models aligning with these goals. Looking back, I guess at least I have been guilty of this many times myself.

So what kinds of modeling goals there are? I argue that there are two main classes of modeling goals, which I refer to as *prediction* and *explanation* (Scholz and Bürkner 2023). With prediction I mean that the goal is to predict future data, that is, quantities that are at least in

[1] In pre-registered studies, the concept of data-based model selection becomes challenging. In such studies, we often have to to fix specific models to fit before even gathering the data. That said, this requirement should not stop us from performing additional analysis with other, potentially better models, once we have gathered the data.

principle observable even if they are not yet available at the time of model building. If prediction is our main goal, we may choose to disregard the model structure (i.e., treat it as a black box) and just focus on how well the model predicts new data (by which I don't want to imply that I recommend disregarding model structure). With explanation I mean that the goal is to understand the latent (i.e., unobservable) state of the world. In this case, we care about the latent parts of the model, that is, its parameters or other latent quantities implied by them. Accordingly, we will seek to build models whose parameters we can argue to represent some latent aspects of the world that we care about.

If our modeling goal is prediction, we will have an easier time with real data. We just train our model on some data and then evaluate it's *predictive performance* on new data, unseen during training. In a nutshell, this is the basic logic of building machine learning models. Their parameters are usually not interpretable and generally of secondary matter. All we care about, is how well the model predicts new data. In a way, this is a conceptually easy perspective as our goal is straightforward and we have a good idea how to practically evaluate it.

If our goal is explanation however, things become conceptually and practically more involved. In reality, model parameters may often not truly "exist" in the real world. And even if they did exist in some sense of the word, we will not know their true value in reality. If we did, why bother building any model to estimate them? Accordingly, doing model comparison that directly targets explanation is not something we can do for real data. We can only study it in the context of simulation studies where we know and fully control the ground truth. There, we can investigate a model's *parameter recovery* that is, how close the posterior of its parameters (or some other parameter-implied quantity of interest) is to the true value we aim to estimate, on average across simulated dataset. For me, parameter recovery is the statistical operationalization of explanation, as the latter is often used in a more vague sense in the philosophy of science literature.

In most chapters of this book, our main goal will be explanation, and prediction will only be of secondary concern. For example, in the epilepsy data, we care primarily about understanding the effect of the epilepsy treatment, which if of course a latent concept that we have no direct access to. At the same time, in most chapters of this book, we will analyse real data. So we cannot infer the explanative capabilities of our models by means of studying their parameter recovery. Instead, we have to look at the models' predictive performance while paying close addition to their structure. The hope is that, ideally, we can use prediction as a proxy for explanation, such that the better predicting models also provide more valid explanation for the latent concept of interest (e.g., the latent treatment effect). I consider this proxy use valid as long as our models are causally consistent, that is, are sufficiently representing the true causal structure of the involved variables. At least, we have found evidence for this perspective in relatively simple models (Scholz and Bürkner 2023) and I am currently unaware of any evidence contradicting it.

While causality is a highly important topic, and still under-appreciated in many sciences, I do not have the space in this book to talk about causality much. For an introduction to causality from a Bayesian perspective, I recommend reading, for example, Chapter 6 of Statistical

Rethinking (McElreath 2019), a book I highly recommend more generally. It even has a brms translation, where all models are fitted with brms instead of with the rethinking package designed to accompany the book natively (Kurz 2019).

I know you are here to learn about practically performing model comparison, but given the complexity of this topic, I need a bit more space for some preparatory thoughts before we can start running code.

2.4 Measuring Predictive Performance

There is not a single "best" predictive metric agnostic to our models and modeling goals. Accordingly, we need to choose metrics that fit to our goals. If possible, that is. In practice I often see a rather different approach. It is common to just apply whatever metrics are currently standard in a given field. And I cannot blame anyone for that. As you will see in this chapter and more generally in this book, I am a bit guilty of the same. I have my favorite methods and apply them consistently to most modeling situations I encounter. Perhaps in some of these situations, my choice of predictive metrics for model comparison is actually not particularly good after all.

As explained above, evaluating and comparing the predictive performance of Bayesian models will be the main focus of this chapter out of practical necessity. To make it easier for you to connect all the topics that will follow into a coherent picture, I will give you a quick overview first. Don't worry if this does not fully make sense yet upon first reading. It will make more sense after you have finished reading this chapter.

First, we distinquish between absolute and relative predictive performance. Absolute predictive metrics *have clearly defined and interpretible optimal value*, indicating "perfect predictions" so to say. That way, a model's predictive performance can be compared to this optimal value without having to resort to comparing multiple models. Relative predictive metrics do not have such an optimal value, at least not an easily interpretable one, so we have to compare them across multiple models to make any sense of the results. Absolute metrics can always used in a relative manner, but not the other way round.

Second, we distinquish between prior and posterior predictive performance. Prior predictive metrics measure how well a model performs *before seeing the current data*, that is just based on the information in prior alone – whatever non-current-data information was used derive the prior in the first place. Posterior predictions measure how well a model performs *after seeing the current data*, that is based on both current data and prior information.

Third, we distinguish between in-sample and out-of-sample predictive performance. We measure in-sample predictive performance if the data we evaluate the model on (the *test data*) is the same (or a subset of) the data on which we have trained the model (the *training data*). Contrarily, we measure out-of-sample performance if the test data does not overlap with the training data. This distinction only makes sense when we care about posterior predictions

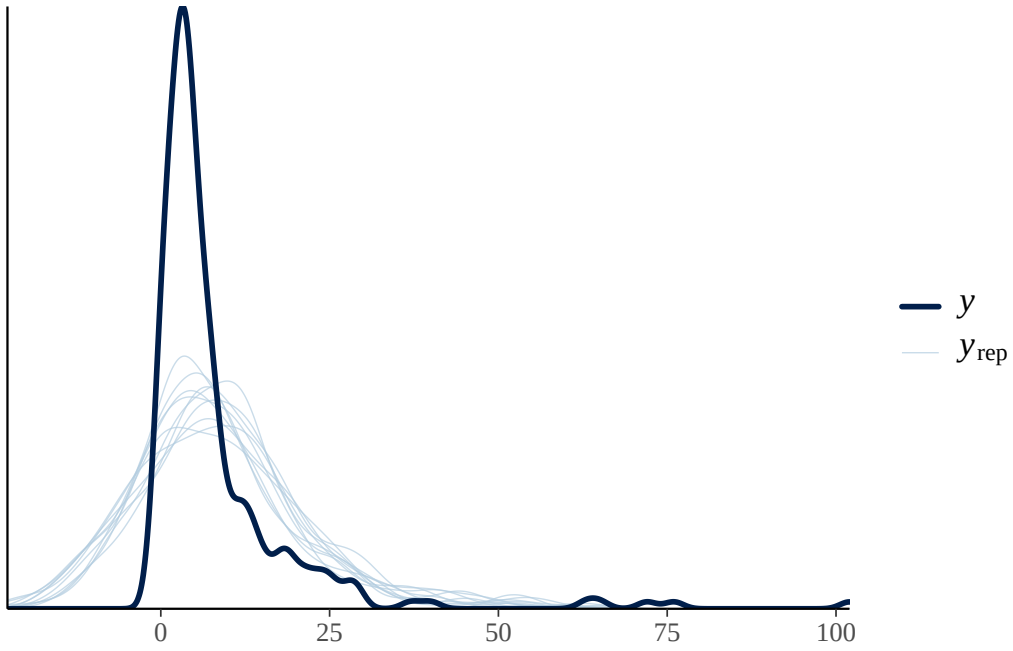
because prior predictions have no training data to begin with. In that sense, prior predictions are always "out-of-sample".

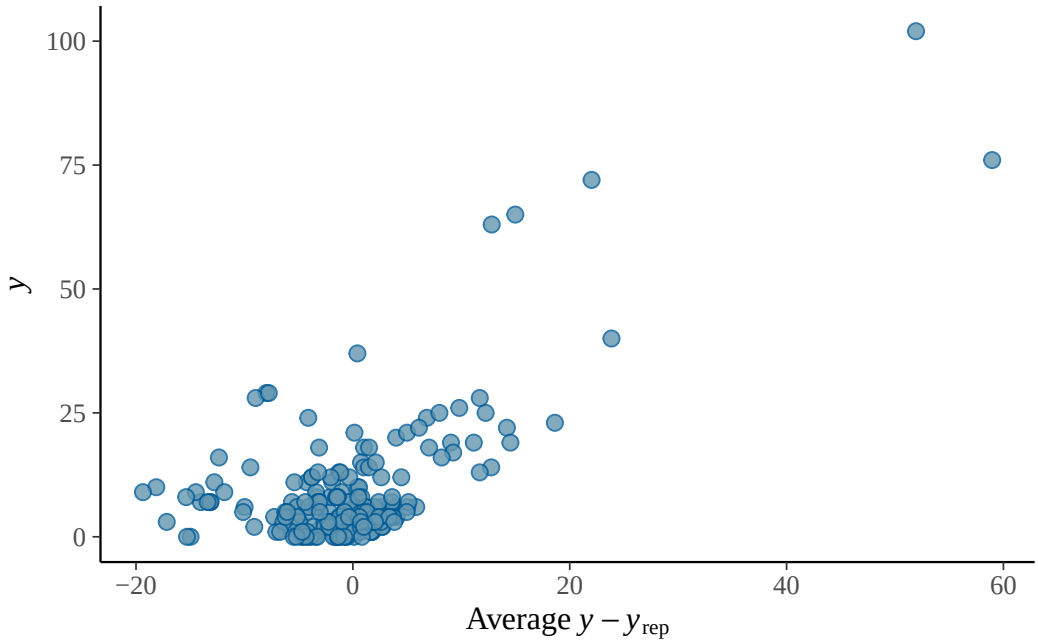
2.5 Absolute predictive performance

Let's start with evaluation of absolute predictive performance that can be performed on the basis of just a single model. Further, let's focus on in-sample predictions for now. We will reuse a bunch of models from the last chapter, thus continuing our case study of modeling the number of epileptic seizures. For more detailed explanation how we came to these models, please see the last chapter. Let's look at a simple Gaussian model first:

```
fit_epi_gaussian2 <- brm(count ~ Trt + Base, data = epilepsy)
```

In the last chapter, we have already seen some basic form of evaluating absolute prediction graphically, by means of posterior predictive checks. Two examples, you can find below:





Different PP-checks can be differently informative in different situations, so it is hard to give general recommendations for this. What I like is to start with the default PP-check (`type = "dens_overlay"`) just to see if things roughly makes sense marginally. Additionally, the `"error_scatter_avg"` type resembling residual plots provides useful information into the error structure of our model. Both plots clearly point to problems in our simple Gaussian model fit to the epilepsy data. For a detailed overview of available PP-check option, I recommend checking out `?bayesplot::PPC`.

2.5.1 Measures of Explained Variance

But what about numerical measures? One measure that may come to mind is the *coefficient of determination*, usually denoted by R^2, which is also often referred to as percentage of explained variance or percentage of explained variation. In its basic form, R^2 is defined only for Gaussian linear models as:

$$R^2_{\text{basic}} = 1 - \frac{\sum_{n=1}^{N}(y_n - \hat{y}_n)^2}{\sum_{n=1}^{N}(y_n - \bar{y})^2}$$

The numerator in the ratio describes the sums-of-squares of the errors, as we compute the difference between observed responses y_n and model-predicted responses $\hat{y}_i$. The denominator normalizes these squared errors on the total variation of the responses around their mean $\bar{y}$.

Accordingly, the ratio describes the percentage of *unexplained variation*; and 1 minus this term is thus a measure of *explained variation*.

The definition of R^2_{basic} is not inherently Bayesian, but we can readily turn it into a fully Bayesian measure by considering posterior draws over the mean of the posterior predictive distribution (`posterior_epred`) as our $\hat{y}_i$. Then, we can compute draws from the posterior of R^2_{basic} as follows:

$$(R^2_{\text{basic}})^{(s)} = 1 - \frac{\sum_{n=1}^{N}(y_n - \hat{y}_n^{(s)})^2}{\sum_{n=1}^{N}(y_n - \bar{y})^2}$$

Since `fit_epi_gaussian2` is just a Bayesian linear regression, we can actually just go ahead and compute (the posterior distribution of) this basic R^2_{basic}:

```
# compute draws of the predictive errors based on posterior_epred
errors <- predictive_error(fit_epi_gaussian2, method = "posterior_epred")
str(errors)
```

```
 num [1:4000, 1:236] 3.35 3.38 3.42 3.36 4.19 ...
```

```
# sum errors over observations
error_variation <- rowSums(errors^2)
str(error_variation)
```

```
 num [1:4000] 15121 15049 15003 15148 15110 ...
```

```
# compute R2_basic
overall_variation <- sum((epilepsy$count - mean(epilepsy$count))^2)
R2_basic_epi_gaussian2 <- 1 - error_variation / overall_variation
posterior_summary(R2_basic_epi_gaussian2)
```

```
      Estimate   Est.Error      Q2.5     Q97.5
[1,] 0.5765227 0.004587625 0.5644022 0.5816325
```

The results tell us that roughly between 56% and 58% variation in the responses is explained by the model, least when considering in-sample fit for now. This is quite substantial and likely primarily driven the the `Base` predictor of baseline epileptic seizures unsurprisingly showing a very strong relation to the number of epileptic seizures during treatment.

If we were to compute a basic linear regression with the `lm` function instead of with `brm` and checked out the obtained R^2, we would see that the point estimate obtained from the former

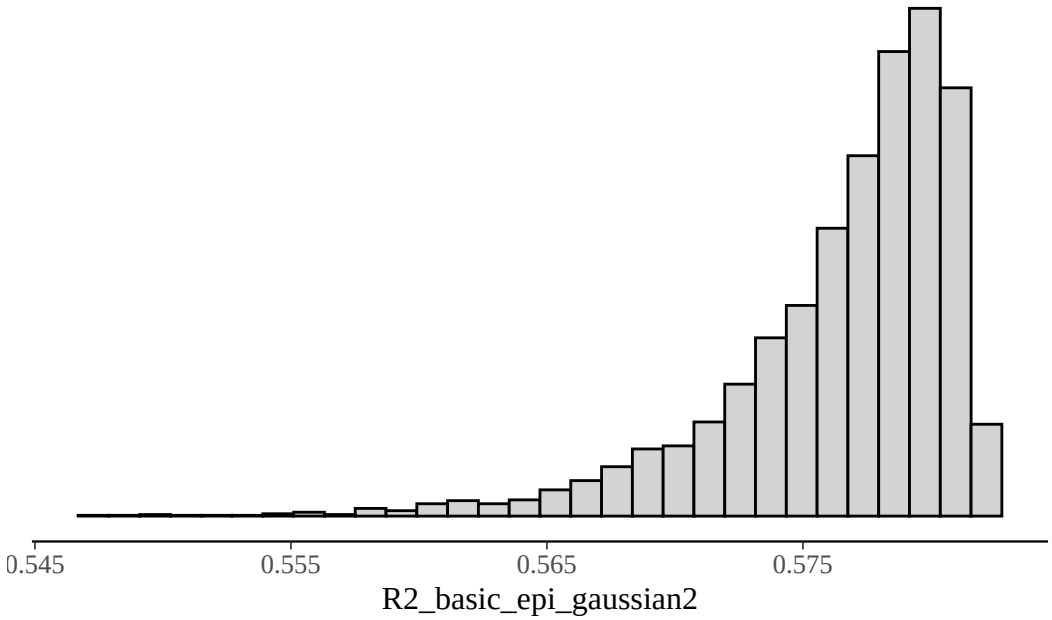


Figure 2.1: Histogram of the posterior of R^2_{basic} for the `fit_epi_gaussian2` model.

would be in the bulk of the posterior distribution obtained from the latter; at least when using weakly-informative priors. Check it out yourself as an exercise.

The R^2_{basic} is a great starting point, but it doesn't readily generalize to models that are much more complicated the Gaussian linear models. In particular it doesn't readily generalize to most- other likelihood families. Accordingly, it makes sense to introduce a more general form of R^2 that we can apply to (almost) all brms models, regardless of what likelihood families they have. A quite straightforward general definition was given by Gelman et al. (2019). Following the intuition about R^2, their measure is based on the ratio of explained variance and the sum of explained and error variance:

$$R^2_{\text{general}} = \frac{\text{explained variance}}{\text{explained variance} + \text{error variance}} = \frac{\text{Var}(\hat{y})}{\text{Var}(\hat{y}) + \text{Var}(\hat{e})}$$

Where $\text{Var}(\hat{y})$ is the variance of the posterior predicted mean over observations (again `posterior_epred`) and $\text{Var}(\hat{e})$ is the variance of the model-implied errors over observations, where $\hat{e}_n = y_n - \hat{y}_n$. Notice that this is the same error definition as was used in R^2_{basic}. Still, even for Gaussian linear models, the two discussed definition of R^2 are not identical, but their interpretation as percentage of explained variance is the same.

We can readily formulate a posterior-draws version of R^2_{general} by computing $\text{Var}(\hat{y})$ and $\text{Var}(\hat{e})$

for each posterior draw, which then implies posterior draws of R^2_{general}:

$$(R^2_{\text{general}})^{(s)} = \frac{\text{Var}(\hat{y})^{(s)}}{\text{Var}(\hat{y})^{(s)} + \text{Var}(\hat{e})^{(s)}}$$

We could code this up by hand, but brms already supports it natively via the `bayes_R2` method. For our Gaussian linear regression model, we get

```
bayes_R2(fit_epi_gaussian2)
```

```
    Estimate  Est.Error      Q2.5    Q97.5
R2 0.5794807 0.02717851 0.5201627 0.627415
```

So, in qualitative accordance with the basic R^2, we see that this model "explains" around 58% of the response's variance. What is more, we are quite certain about this estimate with the 95% CI of the R^2 posterior ranging from 52% to 63%. By default, the output already provides summaries over posterior draws. If we were interested in the draws themselves, we could add argument `summary = FALSE` as is the case in a lot of other post-processing functions being able to output draws or summaries thereof.

How are things looking with our slightly more complicated but still Gaussian linear model involving an additional interaction term between treatment and baseline epilepsy score?

```
fit_epi_gaussian3 <- brm(count ~ Trt * Base, data = epilepsy)
```

```
bayes_R2(fit_epi_gaussian3)
```

```
    Estimate Est.Error     Q2.5     Q97.5
R2 0.6075166  0.025635 0.552121 0.6518007
```

Apparently, at least according to R^2, adding the interaction makes bare any difference and just adds around 2%-3% more explained variance. Additionally noting the strong overlap between R^2 posterior distributions of both model, this difference is unlikely to be substantial in most sensible definitions of that word. That said, I still remember a professor of mine talking about another professor working in neuroscience stating that: "Whenever a model explains for than 1% of the fMRI data's variance, she gets nervous." What we can learn is that small changes (in term of model predictions, R^2, etc.) can mean a lot depending on the context and the signal strength we can reasonably expect to be in our data.

2.5.2 Measures of Squared Errors

Another common measure of predictive performance that finds widespread application in statistics and machine learning is that root mean squared error (RMSE). As with R^2, there is not necessarily one single definitions, but rather several, all following the same intuition of measuring (the root of) an average squared difference between target values and values used to approximate the targets. For example, we can define the RMSE simply as

$$\text{RMSE}^{(s)}_{\text{basic}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{y}_n^{(s)})^2},$$

which you will easily identify as the numerator of R^2_{basic} defined earlier. As such, $\text{RMSE}_{\text{basic}}$ is just the "unnormalized" version of R^2_{basic}. While the latter is scale free and varies between 0 and 1, the former has the same scale as the response y and can vary between 0 and infinity in general. If we care about and understand the scale of y, the RMSE may be more interpretable. For example, in the context of logistics, we are interested, among others, in arrival time predictions of goods. To make sure that our supply chains run smoothly, we would want our arrival time predictions to be as accurate as possible, e.g., not being off by more than a certain number minutes on average, say, 60 minutes. This is something we could easily estimate with an RMSE because we understand the scale of our response (time in minutes) and as such understand the scale of the RMSE. In contrast, an R^2 measure would be a bit useless since we wouldn't know whether or not, say, 80% of explained variance would mean we would reach our desired average prediction error of less than 60 minutes.

When computing $\text{RMSE}^{(s)}_{\text{basic}}$, we can largely reuse code that we already needed for R^2_{basic}:

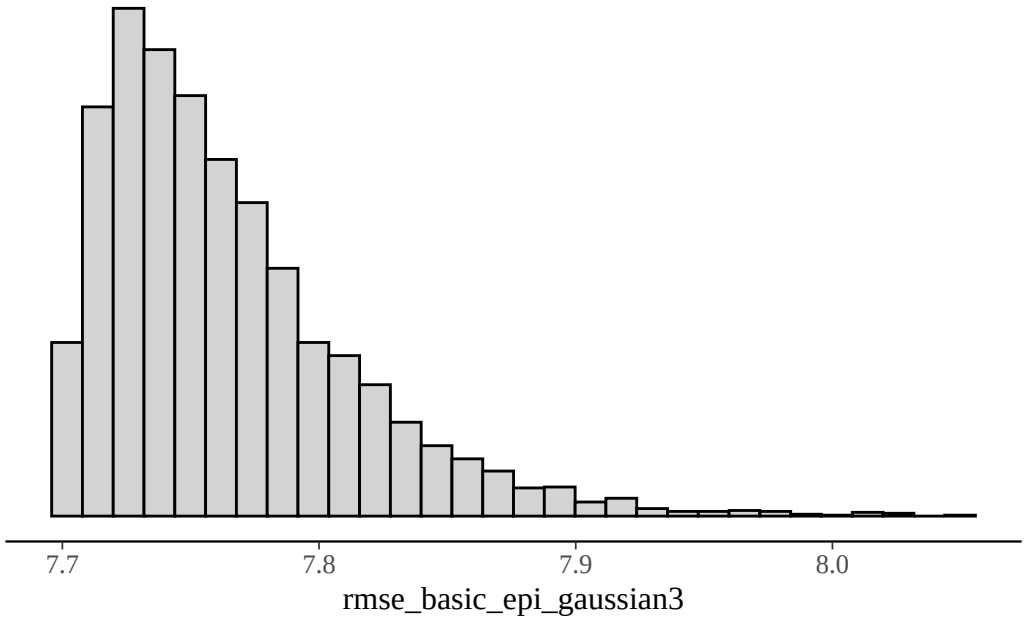
```
# compute draws of the predictive errors based on posterior_epred
errors_epi_gaussian3 <-
  predictive_error(fit_epi_gaussian3, method = "posterior_epred")
str(errors_epi_gaussian3)
```

```
 num [1:4000, 1:236] 1.445 1.068 0.439 1.658 0.546 ...
```

```
# root mean of squared errors over observations
rmse_basic_epi_gaussian3 <- sqrt(rowMeans(errors_epi_gaussian3^2))
str(rmse_basic_epi_gaussian3)
```

```
 num [1:4000] 7.72 7.74 7.8 7.84 7.74 ...
```

```
histogram(rmse_basic_epi_gaussian3)
```



Accordingly, the average posterior predictor error of `fit_epi_gaussian3` is very likely between 7.7 and 7.9 seizure counts, which admittedly is quite a lot, but not surprising given how suboptimal of a model we know all the Gaussian models to be for this data.

Of course, we wouldn't be in statistics if we couldn't also define useful RMSEs in other ways. For example, while $\text{RMSE}_{\text{basic}}$ computes a mean square error of observations n for each posterior draw s, and thus yields a posterior distribution over RMSE values, we could also exchange the use of n and s and compute a mean square error over draws s for each observation n:

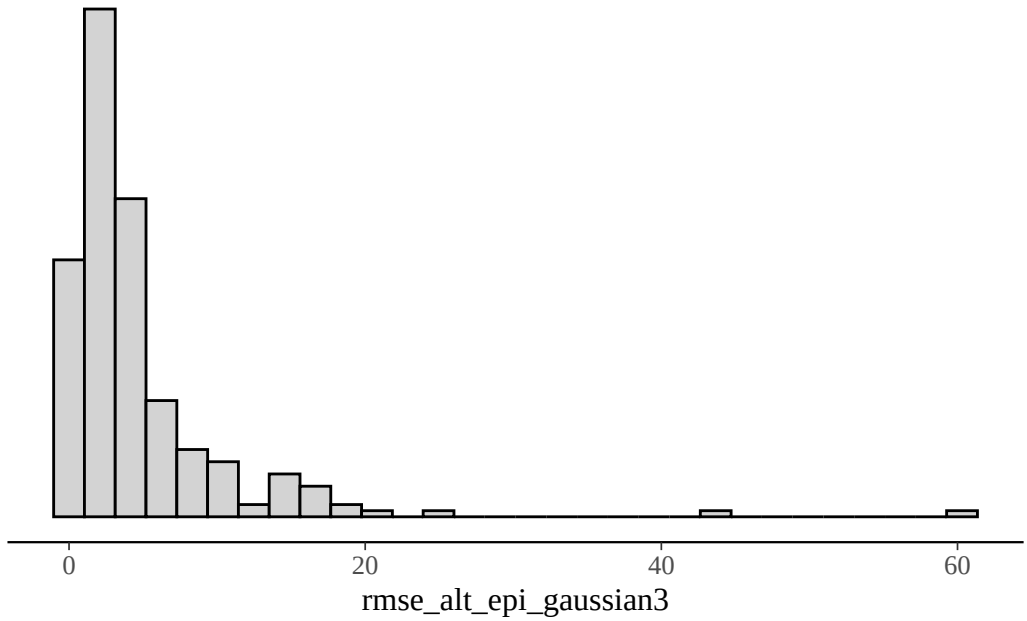
$$\text{RMSE}_{\text{alt},n} = \sqrt{\frac{1}{S}\sum_{s=1}^{S}(y_n - \hat{y}_n^{(s)})^2},$$

In this case, we would get a distribution of RMSE over observations, where each individual RMSE value would be computed over the posterior predictive distribution of a single observation. Both of the above RMSE measures are fully Bayesian as they take into account the uncertainty in the posterior distribution, albeit in different ways.

```
# notice that here we have used colMeans instead of rowMeans as for rmse_basics
rmse_alt_epi_gaussian3 <- sqrt(colMeans(errors_epi_gaussian3^2))
str(rmse_alt_epi_gaussian3)
```

```
 num [1:236] 1.977 0.968 1.032 1.831 10.947 ...
```

```
histogram(rmse_alt_epi_gaussian3)
```



Now, that looks completely different. In the above histogram we see one RMSE value per observation, which clearly tell us the predictions for some observations are extremely bad; up to 60 seizure counts even. We didn't see these extreme results in $\text{RMSE}_{\text{basic}}$ because there we averaged the squared error over observations.

Also, both are not your typical RMSE measure you see, for example, in machine learning papers. There, we typically see only a point estimate $\hat{\tilde{y}}_n$ being used to represent the model-implied predictions, instead of a (posterior) distribution over such predictions for each n. For example, for a Bayesian model this point estimate could simply be the posterior mean $\hat{\tilde{y}}_n = \sum_{s=1}^{S} \hat{y}_n^{(s)}/S$. When using such a point prediction approach, our RMSE definition becomes

$$\text{RMSE}_{\text{point}} = \sqrt{\frac{1}{N}\sum_{n=1}^{N}(y_n - \hat{\tilde{y}}_n)^2}.$$

This is pretty similar to $\text{RMSE}_{\text{basic}}$ except that we have gotten rid of the posterior uncertainty already when computing $\hat{\tilde{y}}_n$.

```
# extract a point estimate of the predictions per observation
ppmean_epi_gaussian3 <- colMeans(posterior_epred(fit_epi_gaussian3))
str(ppmean_epi_gaussian3)
```

```
 num [1:236] 3.26 3.26 1.93 2.46 17.88 ...
```

```r
# compute RMSE based on the responses and point predictions
(rmse_point_epi_gaussian3 <- sqrt(mean((epilepsy$count - ppmean_epi_gaussian3)^2)))
```

```
[1] 7.697822
```

This value is smaller than even the smallest draw from the posterior of $\text{RMSE}_{\text{basic}}$ above, because $\text{RMSE}_{\text{basic}}$ averages over the posterior before computing the sum of squares. This also tells us that comparing, a model from whom we have obtained posterior draws with a model from whom we have obtained only point estimates can be seriously misleading depending on what model comparison metrics we choose to apply. Accordingly, please make sure that all models contain reasonable estimates of posterior uncertainty, that is, are ideally all fitted with the same posterior approximation algorithm (say, with Stan's main MCMC sampler). Otherwise, we may conclude that one model is better in terms of RMSE simply because its output did not contain posterior uncertainty at all, or because it drastically underestimated said uncertainty.

2.6 Relative predictive performance

So far we have only looked at one model at a time, which may, in some cases, be sufficient. But in general, it is more common to compare multiple models against each other and thus investigating their *relative predictive performance*. As a competitor to our Gaussian model, let's consider the corresponding Student-t model from the last chapter:

```r
fit_epi_student1 <- brm(
  count ~ Trt * Base, data = epilepsy, family = student()
)
```

RMSE values can be computed as previously, for example, for RMSE_{alt}:

```r
errors_epi_student1 <-
  predictive_error(fit_epi_student1, method = "posterior_epred")
rmse_alt_epi_student1 <- sqrt(colMeans(errors_epi_student1^2))
```

We can now even compute the pointwise (per-observation) difference in RMSE values:

```r
rmse_alt_diff <- rmse_alt_epi_student1 - rmse_alt_epi_gaussian3
str(rmse_alt_diff)
```

```
 num [1:236] 0.2 -0.645 -0.487 0.124 -0.913 ...
```

This RMSE difference has a mean of -0.36 seizure counts, which isn't that large I would say, in particular in light of the substantial variation in pointwise RMSEs as shown in Figure 2.2. We can attempt to understand the "significance" of the RMSE difference by comparing the mean difference with its corresponding standard error, that is simply computed as the standard deviation divided by the square root of N:

```
se_mean <- function(x) {
  sd(x) / sqrt(length(x))
}
```

```
se_rmse_alt_diff <- se_mean(rmse_alt_diff)
```

We get a standard error of `round(se_rmse_alt_diff, 2)` which is roughly of the same size as the mean difference itself. In another words, the mean difference is only about 1 standard error units away from zero. That is not much. For reference, when assuming a normal distribution and having a not too small sample size, we would call a difference significant (in a frequentist sense) at an α level of 5% if the mean difference is further away from zero than 2 standard error units. So, in our case, we would probably need to have at least twice the current mean difference to start believing in its reliability.

Accordingly, as seen above, we should consider both the difference between models in "raw" RMSE values and in "scaled" RMSE values relative to the corresponding standard error. As we will see in a bit, both is relevant not only for RMSE but also for other metrics of model performance. In our case here, both the raw and the scaled difference in RMSE values is small so, on this basis, we do not see convincing evidence of one model over the other.

So far, we have been looking at RMSEs that were directly related to common R^2 definitions that we introduced before. However, all of them are ignoring the variation induced by the likelihood, that is, do not actually incorporate all the model uncertainties. Remember the difference between `posterior_epred` and `posterior_predict` from Chapter 1? Above, we have always used `posterior_epred`, but nothing stops us to compute the RMSEs with `posterior_predict` as well. This would also be closer to more standard posterior-predictive checks which (almost) all use predictions of new responses rather than predictions on the expected response mean. Also, we would expect the Student-t model to better show its strengths there because of it's ability to deal with outliers.

Let's compute $\text{RMSE}_{\text{alt},n}$ again but compute the errors from the `posterior_predict` predictions:

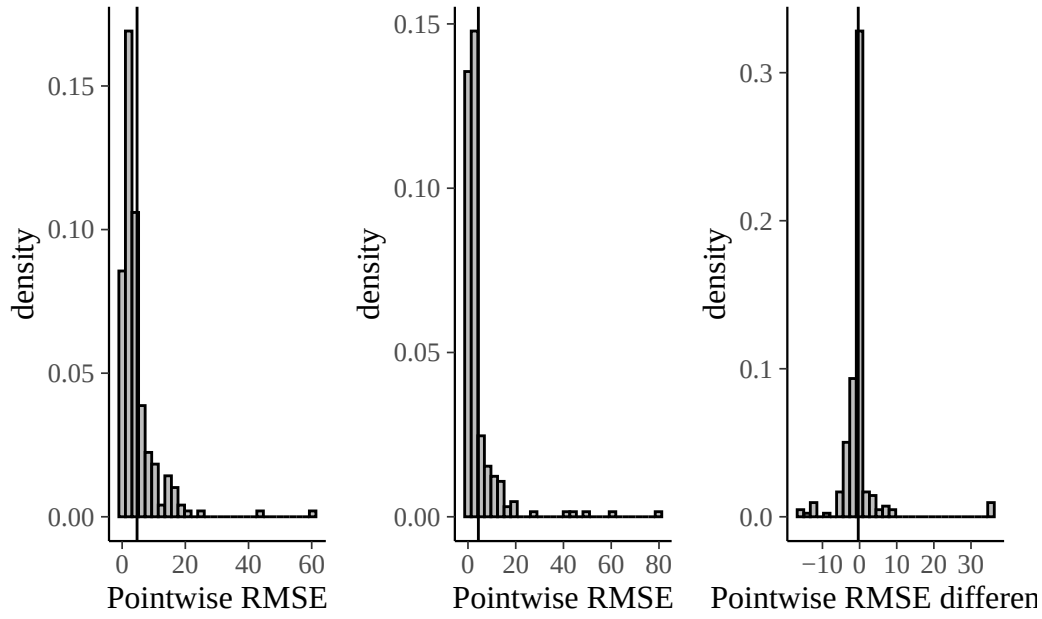
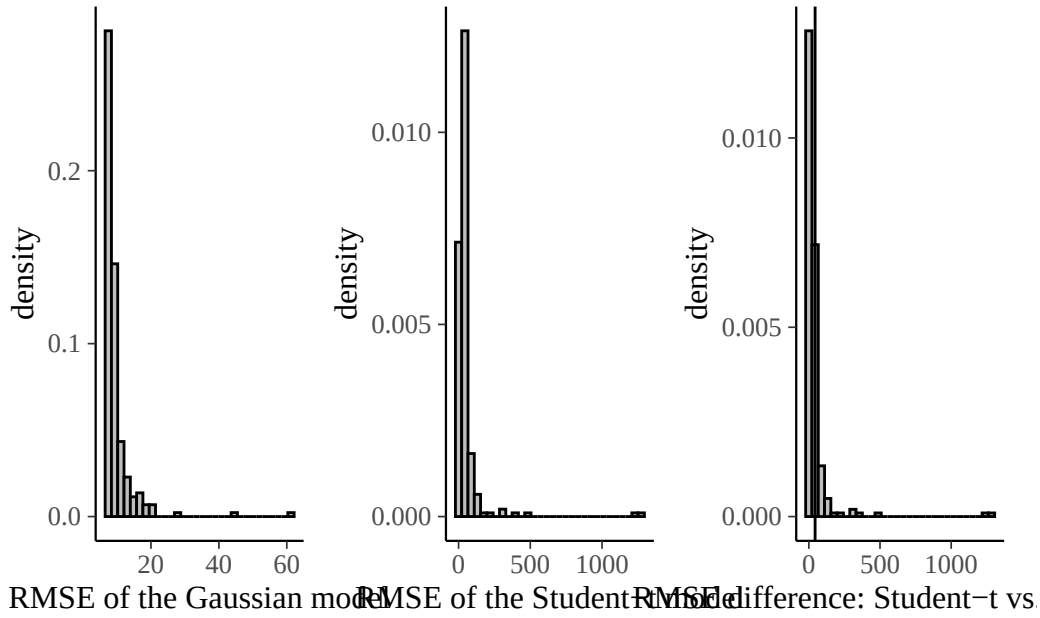


Figure 2.2: Histograms of the pointwise RMSE_{alt} contributions (based on `posterior_epred`) for the Gaussian and Student-t models as well as for their pointwise RMSE_{alt} differences.

```r
# "posterior_predict" is the default method in predictive_error()
errors_pp_epi_gaussian3 <- predictive_error(fit_epi_gaussian3)
rmse_pp_alt_epi_gaussian3 <- sqrt(colMeans(errors_pp_epi_gaussian3^2))

errors_pp_epi_student1 <- predictive_error(fit_epi_student1)
rmse_pp_alt_epi_student1 <- sqrt(colMeans(errors_pp_epi_student1^2))

rmse_pp_alt_diff <- rmse_pp_alt_epi_student1 - rmse_pp_alt_epi_gaussian3
```



RMSE of the Gaussian model RMSE of the Student-t model RMSE difference: Student−t vs.

Figure 2.3: Histograms of the pointwise RMSE_{alt} contributions (based on `posterior_predict`) for the Gaussian and Student-t models as well as for their pointwise RMSE_{alt} differences.

The results are presented in Figure 2.3. We see that, perhaps to our surprise, that the Student-t model has terrible RMSE for some observations; much worse than the normal model. What happened here? Remember the posterior predictive checks of the Student-t model from last chapter whose x-axis we had to restrict in order to see something? The same has happened here: If a Student-t likelihood has small degrees of freedom (as implied here by the data), than the posterior predictive distribution has (no choice but) to produce occasional big outliers. They may then blow of up the scales of plots or strongly influence summary measures that are strongly influencable by outliers such as mean, variance, or standard deviation.

Does that mean the Student-t model is worse than the normal model? As we concluded

already in the last chapter, not necessarily. It just depends on what predictive metric we choose to evaluate. As we had seen above, when using expected posterior predicted values (`posterior_epred`), the Student-t model was even slightly better then the Gaussian model even when including the very high RMSEs of several outliers. This is interesting insofar as, conceptually, the RMSE is very closely related to the normal models. If we were to write down the log-likelihood of the normal model, it looks something like this:

$$\log p(y \mid \theta) = \left(\sum_{n=1}^{N} -\frac{1}{2} \frac{(y_n - \mu_n)^2}{\sigma^2} \right) - N \log(\sqrt{2\pi}\sigma),$$

where μ_n is the predicted mean of observation n and σ is the residual standard deviation parameter assumed constant over observations. If we just focus on the first term and are ignoring the scaling by σ, we see that the log-likelihood of a Gaussian model (also known as *squared loss function* in machine learning) looks very similar to the RMSE. For the latter, we just have the square-root around the sum but that doesn't really change much here. Put differently, a Gaussian model already tries to get the best mean-squared error possible. Accordingly, we can expect Gaussian model to have some advantage in predictive metrics based on mean-squared errors such as the RMSE. That is fine, if we know that minimizing RMSE is what we want from our models. But if we have have no clue what metric we actually care for and just want to have a general purpose metric to compare models "holistically" in some sense, RMSE and close friends may not be ideal. For this reason, we will next learn about more general predictive metrics that are agnostic to the specific model or model structure.

2.6.1 Likelihood Density Scores

Above we have talked all kinds of variations of R^2 and RMSE metrics. In particular, we have seen how the RMSE is related to the log-likelihood of Gaussian models. So what if we just go ahead and use the log-likelihood of models as predictive metric more generally? After all the log-likelihood plays a pivotal role not only in to derive the posterior in Bayesian statistics but also to obtain maximum likelihood estimates in a frequentist framework. Intuitively, the higher the likelihood of the data given the model's parameters estimates (represented as either posterior draws or point estimates), the better the fit of the model to the data. Indeed, many important predictive metrics, Bayesian or otherwise, are based on log-likelihood scores.

Remember from **?@sec-bayesian-inference** that, if the observations $y = (y_1, y_2, \dots, y_N)$ are assumed to be independent conditional on the parameters θ, the likelihood factorizes into a product of pointwise (per-observation) terms:

$$p(y \mid \theta) = p(y_1, y_2, \dots, y_N \mid \theta) = \prod_{n=1}^{N} p(y_n \mid \theta)$$

If we then take the log, the product of the pointwise terms becomes the sum of the pointwise log terms:

$$\log p(y \mid \theta) = \sum_{n=1}^{N} \log p(y_n \mid \theta)$$

Since the log is a strictly monotonic transformation, we are not changing anything fundamental by looking at log scores – at least as long as we are careful. However, we are making a lot of math much simpler by now working with sums instead of products. In particular, this concerns computing gradients because the gradient of a sum is just the sum of the individual (pointwise) gradients. In fact, much of the modern statistics and machine learning relies on this property, so we are working with something quite fundamental here. That said, we can also work with log-likelihood metrics if the likelihood doesn't factorize easily or even at all, but the math becomes more cumbersome in this case (Bürkner, Gabry, and Vehtari 2021).

As a regular user of brms, you do not have to worry about these things, since the package comes with a dedicated `log_lik` method that does all the required math for you. Let's us this method to compare the Gaussian with the Student-t model once more. But first, we take a look at the Gaussian model in isolation:

```
ll_epi_gaussian3 <- log_lik(fit_epi_gaussian3)
str(ll_epi_gaussian3)
```

```
 num [1:4000, 1:236] -2.97 -2.96 -2.98 -3 -3.04 ...
```

The output of `log_lik` has the same structure and `posterior_predict` and friends, that is, it has as many columns as we have observations and as many rows as we posterior draws. Similar to the RMSE_{alt} metric earlier, we will average over posterior draws per observation (to be justified more shortly) such that we obtain one log-likelihood value per observation:

```
llm_epi_gaussian3 <- colMeans(ll_epi_gaussian3)
str(llm_epi_gaussian3)
```

```
 num [1:236] -3 -2.98 -2.98 -3 -3.97 ...
```

We use `colMeans` here to create mean log-likelihood scores per observation, averaged over draws. You can find the corresponding histogram in Figure 2.4. Again, we clearly see two outlying observations that the Gaussian model predicts very badly – remember higher likelihood values are better – but otherwise the scale of the values is hard to interpret. Although log-likelihood values are usually negative, they might also be positive. There is also so particular value that we could generally identify as the "best-possible" log-likelihood. So we generally

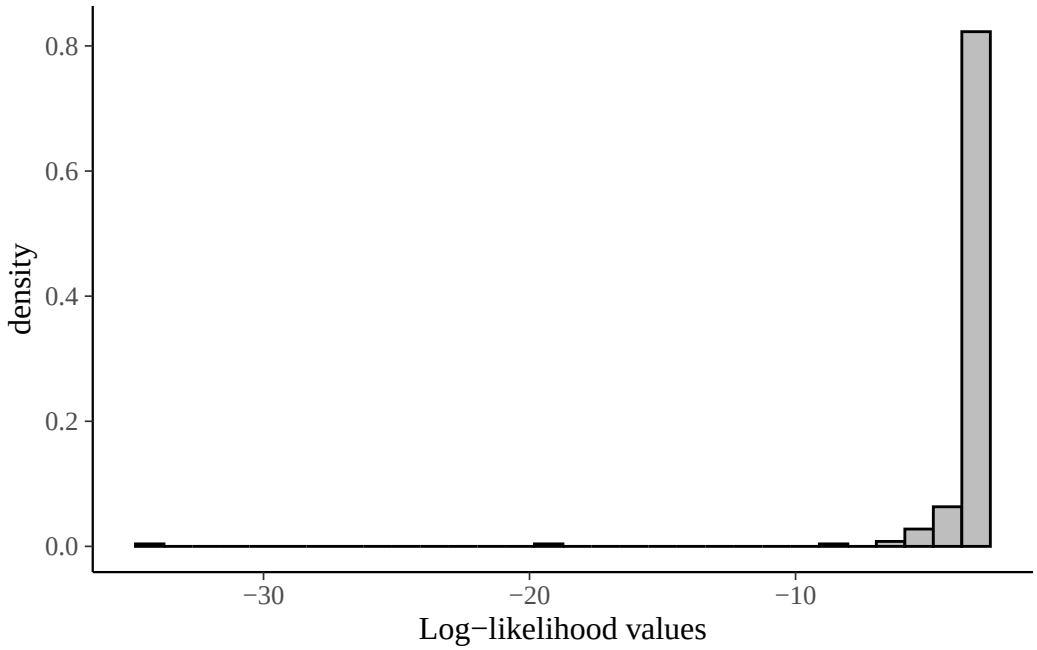


Figure 2.4: Per-observation log-likelihoods of the Gaussian interaction model

need multiple models to compare against in order to make sense of their log-likelihood scores. They are only relative predictive performance metrics.

Let's compute the log-likelihood scores also for the Student-t model and compare them against those of the Gaussian model

```
llm_epi_student1 <- colMeans(log_lik(fit_epi_student1))
llm_epi_diff <- llm_epi_student1 - llm_epi_gaussian3
str(llm_epi_diff)
```

```
 num [1:236] 0.484 1.13 1.095 0.587 -1.273 ...
```

A visualization of the comparison is provided in Figure 2.3. The mean of the log-likelihood differences (vertical black line on the right plot) is slightly positive which points to a slightly better fit of the Student-t model. For reasons, I will explain later on, it is more typical to work with *sums* instead of *means* of log-likelihood values over observations, a quantity that we call LPD:

$$\text{LPD} = \sum_{n=1}^{N} p(y_n \mid y) \approx \sum_{n=1}^{N} \left(\frac{1}{S} \sum_{s=1}^{S} \log p(y_n \mid \theta^{(s)}) \right), \quad \theta \sim p(\theta \mid y)$$

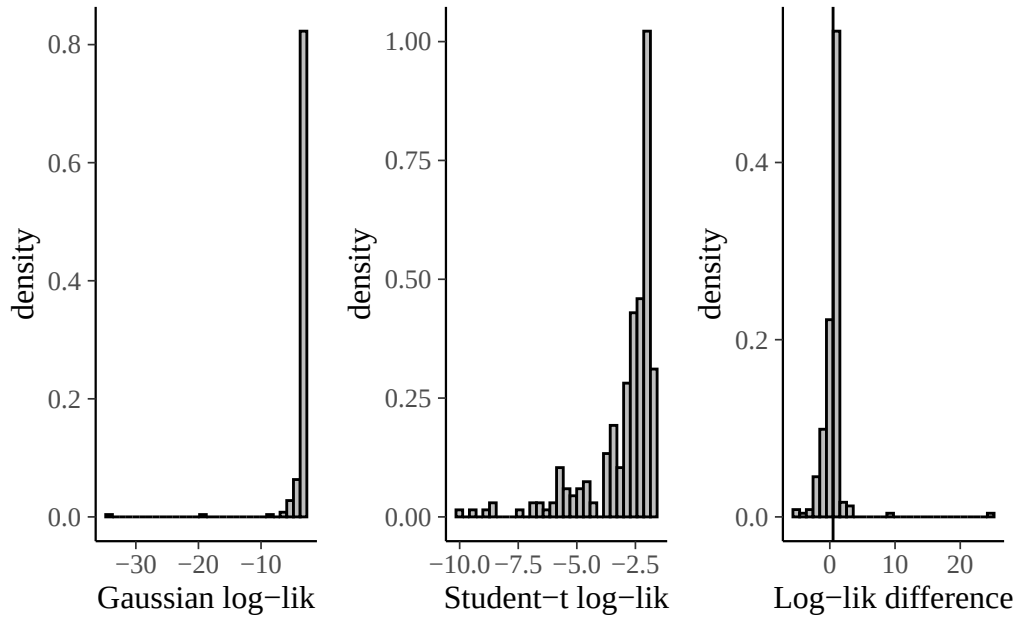


Figure 2.5: Pointwise log-likelihood values of the Gaussian and Student-t interaction models.

So we compute our summary of the LPD differences simply as

```
(lpd_epi_diff <- sum(llm_epi_diff))
```

```
[1] 118.274
```

The corresponding standard error is also not particular difficult to obtain:

```
se_sum <- function(x) {
  sd(x) * sqrt(length(x))
}

(se_lpd_epi_diff <- se_sum(llm_epi_diff))
```

```
[1] 31.05314
```

Accordingly, the Student-t model is clearly better, both in absolute terms (LPD difference of 118 is quite a bit) and in relative terms (about 118/31 $\approx$ 3 SEs away from zero). We had

seen this before in the `posterior_epred`-based RMSE comparisons, but saw the opposite pattern for the `posterior_predict`-based RMSEs. Does that mean the `log_lik` is more similar to `posterior_epred`? Actually, quite the opposite. Both `log_lik` and `posterior_predict` include the aleatoric uncertainty in the likelihood in its values whereas `posterior_epred` does not. However, `log_lik` and `posterior_predict` are doing something fundamentally different with this uncertainty. While `posterior_predict` uses it to *sample* model-implied responses from the posterior predictive distribution, `log_lik` merely *evaluates* the posterior predictive distributions at the observed responses. As a result, if a model has long likelihood tails (e.g., the Student-t model), it will be able to put higher likelihood on outlying observations, thus having higher log-likelihood scores. Since this does not require to sample new responses from the likelihood tails, we do not run into the issue of accidentally creating huge and unrealistic predictions that may strongly affect simulation-based predictive metrics using `posterior_predict`.

When I first encountered this situation, I was quite confused. But it has opened my eyes for all the difficulties and strange things that may happen in model comparison. What you have seen above is my attempt to prepare you mentally for the things you my encounter yourself, or find in papers you read.

Anyway, we cannot not do model comparison regardless of the challenges that may await us. So if I need to make a recommendation that I would argue to use log-likelihood scores as basis for model comparison – unless you have a clear reason to prefer another kind of predictive metric on substantive grounds (Vehtari and Ojanen 2012).

That said, please don't use the basic `log_lik`-based metric above, since we have so far ignored one major aspect that concerns on which data to evaluate model fit: In-sample vs. out-of-sample predictions. So far, we have only evaluated model predictions for observations used to fit the model. So the model had the opportunity to adjust to these very observations that we have used to evaluate its model fit. This is likely to bias model comparison in favor of more complicated models as they are more flexible and can adjust to the specific observations better that they see during training, a process known as *overfitting*. There are many simple examples that can illustrate this problem. For example, when have 3 data points and fit a quadratic model of the form $y_n \sim \text{normal}(b_0 + b_1 x_i + b_2 x_i^2, 1)$, we have 3 parameters (b_0, b_1, b_2) trained on 3 observations (y_1, y_2, y_3). This results in a model that can perfectly predict these three observations, but is likely to generalize poorly to other observations, unseen during training. I won't spell out this example fully here, but if you want to read more, I recommend Chapter 7 of McElreath (2019).

Accordingly, if we don't want to be fooled by overfitting models, it is essentially to evaluate out-of-sample predictive performance to check the models' generalization abilities Vehtari and Ojanen (2012). This is what we will discuss in detail next.

2.7 Out-of-sample predictions

The first question we encounter when trying to evaluate out-of-sample predictions is where to actually get new data to evaluate the model on? In the (usual) absence of any actual new data, we simply go ahead and split our data at hand into two parts, one used for training (the *training data*) and the other we use for model evaluation (the *test data*)[2]. That's easy and readily implemented for our case study. Let's say we choose to use roughly 80% as training data roughly 20% as test data, whereas the assignment of each observation to training or test data is done randomly:

```
set.seed(8973)
epilepsy <- epilepsy %>%
  mutate(use = sample(c("training", "test"), n(), replace = TRUE, prob = c(0.8, 0.2)))
```

```
table(epilepsy$use)
```

```
    test training
      43      193
```

```
epilepsy_train <- epilepsy %>% filter(use == "training")
epilepsy_test <- epilepsy %>% filter(use == "test")
```

We continue to the training data y as before and introduce the test data $\tilde{y}$ to indicate that they are separate from the training data. Using a total of $\tilde{N}$ test observations, we arrive at the out-of-sample version of the LPD, which we will call *expected* LPD, or ELPD in short:

$$\text{ELPD} = \sum_{n=1}^{\tilde{N}} p(\tilde{y}_n \mid y) \approx \sum_{n=1}^{\tilde{N}} \left(\frac{1}{S} \sum_{s=1}^{S} \log p(\tilde{y}_n \mid \theta^{(s)}) \right), \quad \theta \sim p(\theta \mid y)$$

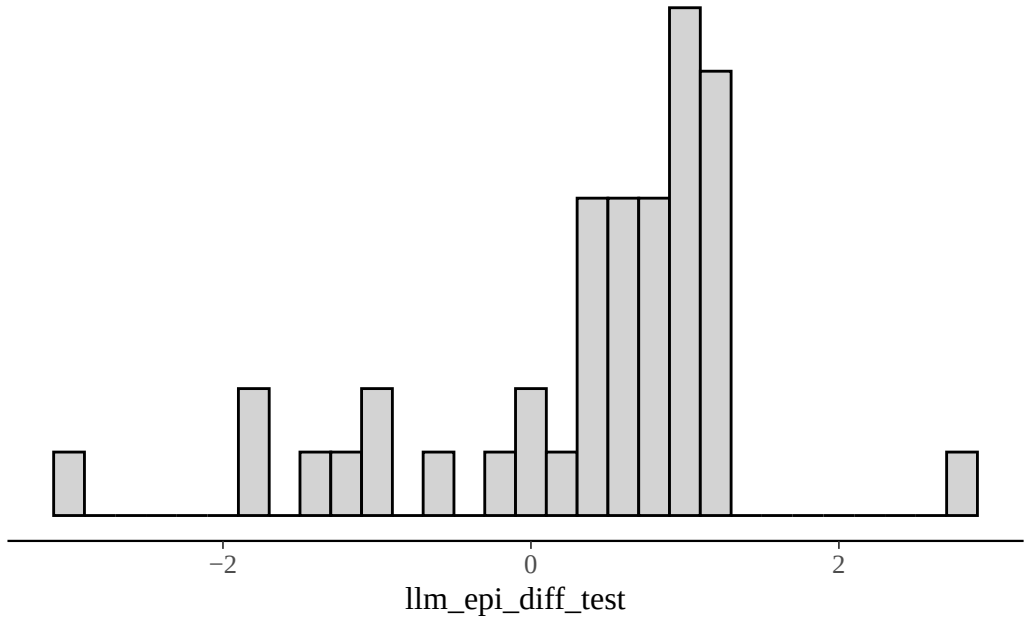
The term "expected" stems from the fact that the above ELPD is an estimate of the log-predictive densities that would be expected if we knew the true data-generating process and could integrate over this process analytically. In some papers, what I call ELPD is simply called LPD or test-data LPD, so it's important to not cling on the specific names but actually check what mathematical definitions are used.

[2] In machine learning workflows, we often further differentiate between *validation data* and (actual) test data. The validation data are used as out-of-sample data to evaluate model fit during the model building procedure. The actual test data are then only used upon the final model evaluation. In machine learning challenges such as those organized by Kaggle (https://www.kaggle.com/), test data are typically hidden from the model builder such that they cannot overfit on them. In that sense, what we call "test data" here in this chapter is perhaps closer to what we would call "validation data" in machine learning.

We are now ready to fit the models on the training data and then evaluate and compare model fit via ELPD on the test data.

```
llm_epi_diff_test <- llm_epi_student1_test - llm_epi_gaussian3_test
```

```
histogram(llm_epi_diff_test)
```




```
(elpd_epi_diff_test <- sum(llm_epi_diff_test))
```

```
[1] 16.6085
```

```
(se_elpd_epi_diff_test <- se_sum(llm_epi_diff_test))
```

```
[1] 7.012555
```

The (out-of-sample) ELPD difference is roughly 17 with an SE of roughly 7. So clearly in favor of the Student-t model. The ELPD difference is noticeably smaller than the (in-sample) LPD difference from earlier, but that is largely due to the fact that we sum over only 20% of observations' log-likelihoods. In fact, if we multiply 17 by 5 to extrapolate to the full size of the data, we end up at roughly 85, which is roughly in the same ballpark than the LPD difference of 117 we found earlier.

Our approach above has one major problem: We rely on an arbitrary split into training and test data. In fact, had we chosen the split differently, for example, just set another seed to the random split we used, we would have gotten a different result. And in a case where the difference between model was smaller, perhaps a qualitatively different conclusion. We see that it isn't wise to rely on a single training-test split, but rather do many of them and then average (or sum) over the results. This is the basic idea of cross-validation. Multiple times split the data into training and test set, each time fit the models on the training data and then evaluate their predictive performance on the test data.

Cross-validation is procedurally easy but computationally brutal. We don't have to fit the model just once but many times, each time on different training data. If fitting the model once is already slow, fitting in many times is often prohibitively slow for practical application. Accordingly, we see to use cross-validation schemes that either have use few training-tests splits or can be approximated efficiently without actually performing any model refits at all. The first leads us to K-fold cross-validation with small K (e.g., $K = 10$). There, we split the data into K subsets, each time leaving out one of the K splits as test data and using the rest as training data. It ensures that all observations appear in a test data set exactly once. You can apply this approach to brms models via method `kfold`.

As long as K is small, we won't need to many model refits. Of course, if our full model based on all data already takes us a few hours, then also $K = 10$ times additional few hours might hurt quite a bit. Accordingly, we will not use K-fold CV as our default approach but rather try to get away with even fewer, ideally no, model refits. This goal is what will lead us to leave-one-out cross-validation approximated via importance sampling approaches, to be discussed in detail next.

2.7.1 Approximate leave-one-out cross-validation

In leave-one-out cross-validation (LOO-CV), we perform N training-test splits, where each time we are leaving out a single observations, fitting the model on the remaining $N - 1$ observations before evaluating model fit on that single left-out observation. If we think about that for a second, it doesn't seem like a wise approach computationally. Usually N is quite large so if we are already unhappy with $K \ll N$ model refits how the hell shall we now fit N models in reasonable time? The reason for choosing LOO-CV is certainly not the many training-tests splits. Instead, we make use of the fact that each of the LOO posteriors, trained on a subset of $N-1$ observations, is actually pretty close to the full posterior, which is trained on all N observations. As a result, we are able to approximate the LOO posteriors with the full posterior and a usually minor adjustment.

Said adjustment can be achieved with *importance sampling*. In a nutshell, importance sampling allows us to approximation expectations (means, variances, etc.) of a target distribution by sampling from another distribution (called the *proposal distributions*) that is similar enough to the target distribution. In the LOO-CV case, the targets are the LOO posteriors and the

proposal is the full posterior. To the full posterior we have access and can sample from since we have fitted the full model. The LOO posteriors we do not have access to unless we are spending considerable amount of computing power and time, which we seek to avoid.

Let me formulate the mathematical idea of importance sampling a bit more generally. If we had access to the target distribution p_t, we could approximate its expectations via samples from p_t itself, as we have done many times in the past chapter, essentially every time we worked with posterior draws. Denoting our parameter-dependent quantity of interest $f(\theta)$ for which we seek to approximate the expectation over the target, this can be expressed as follows:

$$\mathbb{E}_{p_t}(f(\theta)) \approx \frac{1}{S}\sum_{s=1}^{S} f(\theta^{(s)}), \quad \theta \sim p_t(\theta)$$

Now, we can actually choose to sample from our proposal p_p instead of from the target p_t and still approximate expectations over p_t. To correct for the difference between the two distributions all we have to to is replace the simple empirical mean above with a *weighted* mean:

$$\mathbb{E}_{p_t}(f(\theta)) \approx \frac{1}{S}\sum_{s=1}^{S} f(\theta^{(s)})\, w(\theta^{(s)}), \quad \theta \sim p_p(\theta)$$

The challenge, is of course to get the weights $w(\theta^{(s)})$ right. I turns out that the appropriate approach is to first compute values proportional to the density ratio of target and proposal distribution:

$$r(\theta^{(s)}) \propto \frac{p_t(\theta^{(s)})}{p_p(\theta^{(s)})}$$

and then normalizing the ratios such that the obtained weights sum to 1:

$$w(\theta^{(s)}) = \frac{r(\theta^{(s)})}{\sum_{s'=1}^{S} r(\theta^{(s')})}$$

The fact that the importance ratios only need to be *proportional* to the density ratio is highly important if we apply importance sampling to LOO-CV. There, the target is a LOO posterior $p(\theta|y_{-n})$ and the proposal is the full posterior $p(\theta|y)$. If we plug that into the density ratio, expand the terms following from Bayes Theorem and subsequently simplify, we get

$$\frac{p_t(\theta^{(s)})}{p_p(\theta^{(s)})} = \frac{1}{p(y_n \mid \theta^{(s)})} \times \frac{p(y_{-n})}{p(y)}.$$

The former term results from the fact that both the prior and the likelihood contributions of all observations except for the nth appears in both models and thus cancel out. The latter term is the ratio of the marginal likelihood of both models. As we know from **?@sec-bayesian-inference**, marginal likelihoods are notoriously hard to estimate so we try to sneak around them whenever possible. Fortunately, the marginal likelihoods are by definition independent of the parameters, and since we only values proportional to the density ratios, we can simply set the importance ratios to

$$r(\theta^{(s)}) = \frac{1}{p(y_n \mid \theta^{(s)})}$$

Here, the closeness of the proposal and target distribution is reflected in remarkable simple importance ratios.

2.7.2 Pareto smoothed importance sampling

We may stop here and just use the thus obtainable importance weights in the ELPD approximation. However, without further adjustments, (a) the thus obtained approximations may be very noisy and (b) we don't know if we can trust the importance sampling approximations in the first place. As a remedy for both, Vehtari et al. (2024) developed Pareto-smoothed importance sampling (PSIS), where the values of the largest percent of importance ratios (usually the largest 15%) are replaced by the respective quantiles of the generalized Pareto distribution, a common extreme value distribution.

By means of this approach, the largest importance ratios are smoothed, thus reducing the noise in the importance sampling approximation. What is more, the generalized Pareto distribution has a shape parameter k, which indicates how fat the tail of the distribution is, although it is very hard to visually grasp the subtle but mathematically important differences caused by varying k (see Figure 2.6). During PSIS, we get an estimate $\hat{k}$ of this Pareto k parameter. Jumping to conclusions and simplifying a bit: As long as $\hat{k} < 0.7$ we can trust our PSIS approximation of the corresponding LOO posterior, as long as our number of draws S is in the usual range of a couple of thousands.

The explanation for the choice of 0.7 as threshold is actually pretty involved. Below, I attempt a high-level summary (see Vehtari et al. 2024 if you want to understand all the details). The higher the k parameter, the fatter the tail of the Pareto distribution, which has important consequences for the distribution's moments (mean, variance, etc.): The fatter the tails of a distribution the more moments will be *infinite*. In the language of probability theory, we say a moment *exists* if its absolute value is finite. Existence of moments is quite a strange mathematical concept, but for our purposes it is sufficient to just understanding its practical consequences. Suppose we want to get an estimate of the central tendency of a variable θ using the empirical mean $\bar{\theta} = \sum_{s=1}^{S} \theta_s / S$ over a set of S random draws from the distribution $p(\theta)$. This $\bar{\theta}$ estimate can only ever be a good approximation of the distribution's central tendency

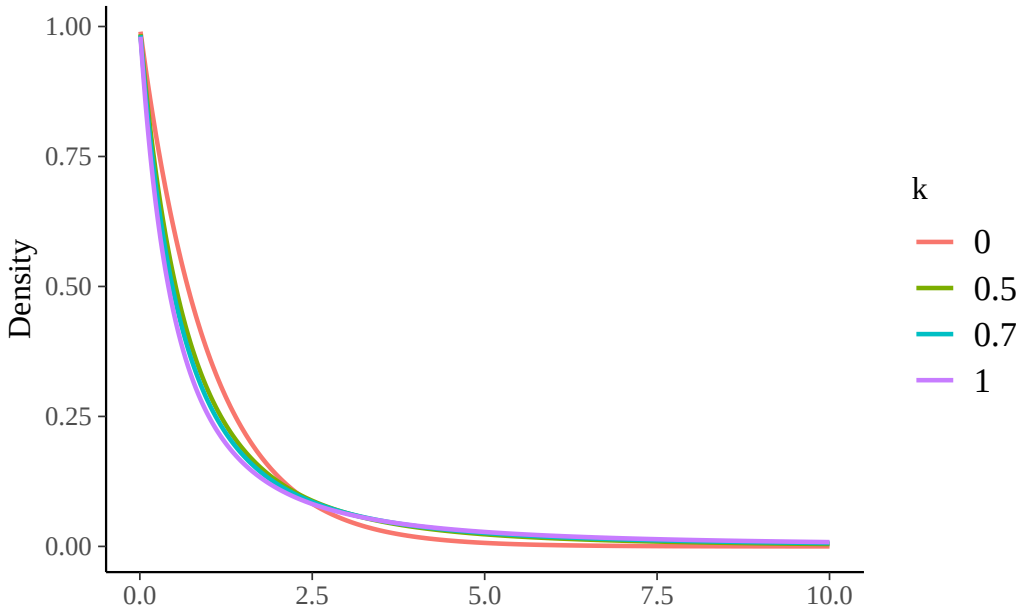


Figure 2.6: Exemplary densities of the generalized Pareto distribution for varying shape parameter k.

if the expected value of $p(\theta)$ (i.e., its true mean) is finite. In other words, if the true mean (or any other moment for that matter) is infinite, their sampling based approximations will be hopelessly bad in helping us learn something about the underlying distribution $p(\theta)$, no matter how many random draws we use.

Back to PSIS and the Pareto distribution, as long as the Pareto k parameter is smaller than 0.5, both mean and variance exist, which means that any sampling-based approximation of the mean is sensible and will converge quickly to the true value. If k is greater than 0.5 but smaller than 1, the mean still exist but the variance does not. This means that for large enough S, we will eventually get a good mean estimate, but the S required for this purpose may be very very large. Fortunately, as long as $k < 0.7$, the sampling-based mean approximation will still converge sufficiently quickly. That is, for the usual number of draws we are working with, having a $k < 0.7$ will be good enough. For $k > 0.7$, convergences becomes too slow, such that we start needing ridiculous number of draws too achieve acceptable mean approximations. Accordingly, it makes sense to use $\hat{k} < 0.7$ as a diagnostic of the reliability of PSIS and importance sampling more generally. The threshold can be refined even further based on S, but for our purposes here, using the $\hat{k} < 0.7$ rule will be good enough.

The complete PSIS LOO-CV procedure is implemented in the loo package, which also provides more details on the $\hat{k}$ threshold choice in `?"pareto-k-diagnostic"`. To perform PSIS LOO-CV on a brms model, all we have to do is run the `loo` method, for example:

```
(loo_epi_gaussian3 <- loo(fit_epi_gaussian3))
```

```
Warning: Found 2 observations with a pareto_k > 0.7 in model
'fit_epi_gaussian3'. We recommend to set 'moment_match = TRUE' in order to
perform moment matching for problematic observations.


Computed from 4000 by 236 log-likelihood matrix.

         Estimate   SE
elpd_loo   -830.7 42.4
p_loo        20.1 11.3
looic      1661.3 84.8
------
MCSE of elpd_loo is NA.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.6, 1.1]).

Pareto k diagnostic values:
                         Count Pct.    Min. ESS
(-Inf, 0.7]   (good)     234   99.2%   1461
   (0.7, 1]   (bad)        0    0.0%   <NA>
   (1, Inf)   (very bad)   2    0.8%   <NA>
See help('pareto-k-diagnostic') for details.
```

We get a warning about two influential observations and a recommendation ("perform moment matching") what to do next. We will ignore this for now but come back to the recommendation in Section 2.7.3.

Let's go through the output in detail. On the very top, we see a short overview of the input data, namely a matrix of log-likelihood values computed for 4000 draws and 236 targets, that is, for our $N = 236$ LOO posteriors. In the table right below we see summaries for three measures. `elpd_loo` is exactly the quantity we defined in Equation @. The higher the ELPD, the better the model fit. `p_loo` is a measure of the *effective number of parameters*, a quite beautiful concept that I will elaborate on a bit later. Finally, `looic` provides an *information criterion* based on LOO-CV. It is just `-2 * elpd_loo` so doesn't really contain any additional information. Yet, it is shown because many people are used to interpreting information criteria such as the Akaike Information Criterion (AIC), so these kind of metrics are familiar for them. Please keep in mind that different from `elpd` *smaller* `looic` values now indicate better fit. As for the summaries, in the first column, we see that point estimate of the three measures, followed by an approximation of their frequentist standard error, just as we had earlier computed manually for the RMSE. Right under the table, we see information on the

Markov Standard Error (MCSE), which tells us how much variation in `elpd_loo` is likely to be caused by the fact the we only used a finite number of posterior draws. For reasons that will become clear in just a bit, we got an `NA` here, so we don't know about the MCSE in this case. At the bottom of the output, in another table, we see information on the Pareto k diagnostics introduced above. We see that 234 LOO posteriors could well be approximated via our PSIS procedure as indicated by their $\hat{k}$ values smaller than 0.7. However, two of the LOO posteriors show extremely bad $\hat{k}$ with values above 1. Apparently, for these two observations, the LOO posterior is too far away from the full posterior for PSIS to bridge the gap. In other words, those two are influential observations whose inclusion vs. exclusion substantially changes the resulting posterior. If you have read the chapter in full until here, you will easily guess which two observations those are. As an immediate implication of these high $\hat{k}$ values, we cannot trust their ELPD contributions as estimated via PSIS (we also didn't get an MCSE estimate for this reason). As a result, we should be careful in trusting the whole PSIS LOO-CV results of the Gaussian model for now, until we have further evaluated the situation. Let's see how LOO-CV does for the Student-t model:

```
(loo_epi_student1 <- loo(fit_epi_student1))
```

```
Computed from 4000 by 236 log-likelihood matrix.

         Estimate   SE
elpd_loo   -704.5 23.5
p_loo         7.2  0.6
looic      1408.9 46.9
------
MCSE of elpd_loo is 0.1.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.5, 1.2]).

All Pareto k estimates are good (k < 0.7).
See help('pareto-k-diagnostic') for details.
```

In terms of $\hat{k}$ values everything seems good now. Apparently, and unsurprisingly given the nature of the Student-t tails, the two observations in question are no longer strongly influential of the posterior. Now is a good time to talk about `p_loo`, which I told you was a measure of the "effective number of parameters". It has its name from the fact that, for well-behaved models with wide priors, `p_loo` roughly equals the total number of parameters. In a way, the number of parameters signals the flexibility of a model. As we increase the informativeness of the priors, the flexibility of the model decreases, so *effectively* we have the flexibility equal to a model with fewer parameters. Accordingly, the effective number of parameter will become smaller than the total number of parameters as we increase the amount of prior information. We

will see this in action especially when we starting talking about multilevel models in Chapter MULTILEVEL.

What is more, when eyeballing the difference between the LOO-CV results for the Gaussian and Student-t model, the latter seems to be clearly better. We can also see this via the appropriate method to compare LOO-CV results:

```
loo_compare(loo_epi_gaussian3, loo_epi_student1)
```

```
                  elpd_diff se_diff
fit_epi_student1     0.0       0.0
fit_epi_gaussian3 -126.2      36.4
```

In `loo_compare` models are ordered from best to worst and all compared to the best model within the set of compared models. It follows that the first row always has zero values since the best model is compared against itself. The first column shows the difference in ELPD values, while the second one shows the SE of this difference. It is kind of magic that this whole procedure allows us not only to obtains error estimates of the individual ELPD values but also of their differences. This SE is not without problems though as shown in Sivula et al. (2022). In particular, if the ELPD differences are close to zero, the SE difference estimates may be biased. Here, our ELPD difference is quite substantial with a values of -126, so the corresponding `se_diff` of about 36 is probably fine. Through these results, we not only see that the Student-t model is clearly better in absolut but also in relative terms, since the ELPD difference is about a factor $126/36 = 3.5$ SE units away from zero.

Alternatively to computing the loo objects separately and then running `loo_compare`, we can also do the whole process within on step via:

```
loo(fit_epi_gaussian3, fit_epi_student1)
```

Let's add the Skew normal model to our comparisons, which we had introduced in Chapter 1:

```
fit_epi_skew_normal1 <- brm(
  count ~ Trt * Base, data = epilepsy, family = skew_normal()
)
```

The PSIS approximation to LOO-CV works a bit better for the Skew normal model than for the Gaussian model. Only one observation is diagnosted as influencial and also not as badly so:

```
(loo_epi_skew_normal1 <- loo(fit_epi_skew_normal1))
```

```
Warning: Found 1 observations with a pareto_k > 0.7 in model
'fit_epi_skew_normal1'. We recommend to set 'moment_match = TRUE' in order to
perform moment matching for problematic observations.


Computed from 4000 by 236 log-likelihood matrix.

         Estimate   SE
elpd_loo   -794.0 32.9
p_loo        16.6  6.1
looic      1588.0 65.8
------
MCSE of elpd_loo is NA.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.4, 1.2]).

Pareto k diagnostic values:
                         Count Pct.    Min. ESS
(-Inf, 0.7]   (good)     235   99.6%   91
   (0.7, 1]   (bad)        1    0.4%   <NA>
   (1, Inf)   (very bad)   0    0.0%   <NA>
See help('pareto-k-diagnostic') for details.
```

If we choose to trust these results for the moment, our model comparison reveals:

```
loo_compare(loo_epi_gaussian3, loo_epi_student1, loo_epi_skew_normal1)
```

```
                     elpd_diff se_diff
fit_epi_student1        0.0       0.0
fit_epi_skew_normal1  -89.5      24.1
fit_epi_gaussian3    -126.2      36.4
```

Clearly, Student-t is much better than either Gaussian or Skew normal. But if we care about the difference between the latter two, the above output only gives us indirect evidence since, in `loo_compare` all models are always compared directly only to the best model. So let's remove the Student-t model from the comparison just to see the difference of the other two models:

```
loo_compare(loo_epi_gaussian3, loo_epi_skew_normal1)
```

```
                     elpd_diff se_diff
fit_epi_skew_normal1   0.0       0.0
fit_epi_gaussian3    -36.7      20.1
```

Apparently, the Skew normal model is better than Gaussian with a sizable absolute ELPD difference and about 2 SE units of relative difference.

In the above code snippets, we have avoided the recomputation of PSIS LOO-CV by storing the `loo` objects directly. This has the disadvantage that they are now independent objects of the fitted model they belong to, so we need to take care of proper naming etc. to make sure there are clearly marked as related. As a convenient alternative, brms offers to store model fit criteria directly inside the fitted model object via the `add_criterion` function. For example, to store LOO-CV results, we can run:

```
fit_epi_gaussian3 <- add_criterion(fit_epi_gaussian3, "loo")
```

```
Warning: Found 2 observations with a pareto_k > 0.7 in model
'fit_epi_gaussian3'. We recommend to set 'moment_match = TRUE' in order to
perform moment matching for problematic observations.
```

Then, when we call things like `loo(fit_epi_gaussian3)`, the previously stored `loo` object is automatically retrieved and recomputation is avoided. That said, if you pass additional arguments to `loo` that may change the result, brms will take it safe and recompute PSIS LOO-CV to provide you with the correct output actually belonging to your input and not to the input you used when running `add_criterion` earlier. For example, if we want to get the importance weights used in the PSIS procedure, we need to save the `psis` object. By default, this is not done such that `loo(fit_epi_gaussian3)$psis_object` returns `NULL`. If we now add `save_psis = TRUE`, brms recognizes the new argument and recomputes LOO-CV instead of retrieving the previously stored object:

```
loo(fit_epi_gaussian3, save_psis = TRUE)$psis_object
```

```
Recomputing 'loo' for model 'fit_epi_gaussian3'
```

```
Warning: Found 2 observations with a pareto_k > 0.7 in model
'fit_epi_gaussian3'. We recommend to set 'moment_match = TRUE' in order to
perform moment matching for problematic observations.
```

```
Computed from 4000 by 236 log-weights matrix.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.6, 1.1]).
Pareto k diagnostic values:
                         Count Pct.    Min. ESS
(-Inf, 0.7]   (good)     234   99.2%   1461
   (0.7, 1]   (bad)        0    0.0%   <NA>
   (1, Inf)   (very bad)   2    0.8%   <NA>
See help('pareto-k-diagnostic') for details.
```

2.7.3 Correcting the PSIS approximation

The LOO-CV approximations of both the Gaussian and the skew normal model where flagged as potentially unreliable because of a few observations with high $\hat{k}$ values. The ELPD differences with the Student-t model are unlikely to be explained by this unreliability, simple because the differences are so large and the influential observations too few. So if you encounter such a situation in practice, you may not worry further if some of your clearly worse predicting models have a few influential observations. For other cases, especially when ELPD differences are much smaller, or when all models have a non-trivial number of influential observations, you need to be highly careful in interpreting the PSIS LOO-CV results. Below, I will show you how to improve the trustworthiness of these estimates.

The first, and perhaps most simple thing we can as a remedy is to actually obtain the exact LOO posteriors of the influential observations in question, by refitting the model each time leaving out one such observation. This is of course computationally expensive, but if we just have to do it a few times, it is often better than performing full on K-fold-CV. In brms, this is implemented via the `reloo` method which requires both the fitted model and a corresponding `loo` object. Alternatively, if have have previously stored the `loo` object inside the model via `add_criterion`, just passing the model is sufficient. Here, we specify both objects explicitly:

```
(reloo_epi_gaussian3 <- reloo(fit_epi_gaussian3, loo = loo_epi_gaussian3))
```

```
Computed from 4000 by 236 log-likelihood matrix.

         Estimate   SE
elpd_loo   -831.8 43.0
p_loo        21.2 12.0
looic      1663.6 86.0
------
MCSE of elpd_loo is 0.4.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.6, 1.1]).
```

```
All Pareto k estimates are good (k < 0.7).
See help('pareto-k-diagnostic') for details.
```

If we, for some reason, already know at the time of running `loo` in the first place that we want to perform `reloo` on influential observation, we may alternatively also specify this option directly in the `loo` method:

```
reloo_epi_gaussian3 <- loo(fit_epi_gaussian3, reloo = TRUE)
```

Fitting the two LOO models took some seconds but reliably got rid of the high $\hat{k}$ warnings as we sampled from the LOO posteriors in question directly via MCMC. The ELPD value as well as all other displayed LOO-CV summaries have changed little from before to after `reloo`:

```
loo_compare(reloo_epi_gaussian3, loo_epi_gaussian3)
```

```
                 elpd_diff se_diff
fit_epi_gaussian3  0.0       0.0
fit_epi_gaussian3 -1.1       0.9
```

In particular, the changes in the ELPD estimates are much smaller than and ELPD difference between models we have seen earlier. Of course, this may not always be the case, so take the Pareto k warnings of PSIS LOO-CV seriously. For completeness, performing `reloo` also for the skew normal models leads to little change there either.

```
reloo_epi_skew_normal1 <- reloo(fit_epi_skew_normal1, loo = loo_epi_skew_normal1)
```

Unsurprisingly, after seeing so little change through `reloo`, comparing the rectified LOO-CV results of the three models leads to no qualitative change in conclusion that the Student-t model is to be preferred:

```
loo_compare(reloo_epi_gaussian3, loo_epi_student1, reloo_epi_skew_normal1)
```

```
                    elpd_diff se_diff
fit_epi_student1       0.0       0.0
fit_epi_skew_normal1  -89.6      24.2
fit_epi_gaussian3    -127.3      36.9
```

Using `reloo` is alright if the individual models don't take too long to fit but of course becomes daunting otherwise. As an alternative, we developed a method termed *moment matching*, which extends the range of importance sampling without requiring model refits (Paananen et al. 2021). Without going into the details, moment matching allows to shift and scale the proposal distribution to be closer to the target before performing importance sampling. As a result, the distance between the proposal and the target that can be bridged with importance sampling becomes substantially larger. In brms, we can activate moment matching during `loo` computation via

```
(mmloo_epi_gaussian3 <- loo_moment_match(
  fit_epi_gaussian3, loo = loo_epi_gaussian3
))
```

```
Computed from 4000 by 236 log-likelihood matrix.

         Estimate   SE
elpd_loo   -831.6 42.9
p_loo        21.0 11.9
looic      1663.2 85.7
------
MCSE of elpd_loo is 0.1.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.6, 1.1]).

All Pareto k estimates are good (k < 0.7).
See help('pareto-k-diagnostic') for details.
```

Also moment matching succeeded in providing good estimates for the two previously problematic LOO posterior and leads to practically identical results as `reloo`:

```
loo_compare(loo_epi_gaussian3, mmloo_epi_gaussian3, reloo_epi_gaussian3)
```

```
                  elpd_diff se_diff
fit_epi_gaussian3  0.0       0.0
fit_epi_gaussian3 -0.9       0.8
fit_epi_gaussian3 -1.1       0.9
```

Due to the implementation details of moment matching, the method requires posterior draws of all variables defined in Stan's parameters block. This is automatically fulfilled for many simple brms models. But for more complex models, brms does not store all such variables by default, because they are otherwise irrelevant for most-processing and just blow up the model

sizes. However, as a result, moment matching may fail, in which case it will suggest you to set `save_pars = save_pars(all = TRUE)` in `brm`. If you initially forgot to set this option and want to apply moment matching, you will have to refit the full model with said option being activated. A bit annoying, I know. Perhaps in the future I will come up with a more convenient and efficient way to handle this case.

Let me finish this section by comparing the `reloo` with the `moment_match` option. If moment matching succeeds in reducing the $\hat{k}$ below 0.7, I consider it preferable over `reloo`. Not only because it is usually faster, but because – perhaps counter-intuitively – it may sometimes even be more accurate than the brute force `reloo` approach. If you are interested in the details, check out our moment matching paper (Paananen et al. 2021).

2.7.4 Leave-one-out R^2

We have seen how log scores can be used to form predictive metrics both to evaluate in-sample and out-of-sample fit. The same options exist for R^2 and RMSE metrics too. In order to obtain out-of-sample R^2 and RMSE metrics, we simply have to perform predictions using the posterior obtained from the training data and compare it against test data observations, unseen by the model during training. Take R^2_{basic} as an example, where we had original drawn predictions from the full model's posterior predictive distribution thus evaluating in-sample fit:

$$(R^2_{\text{basic}})^{(s)} = 1 - \frac{\sum_{n=1}^{N}(y_n - \hat{y}_n^{(s)})^2}{\sum_{n=1}^{N}(y_n - \bar{y})^2}, \quad \hat{y}_n^{(s)} \sim p(y_n \mid y)$$

We can easily create, say, a LOO-CV version of R^2_{basic} by drawing $\hat{y}_n^{(s)}$ from the LOO posterior predictive distribution of the ith data point: $\hat{y}_n^{(s)} \sim p(y_n \mid y_{-n})$ instead. And as with the log scores, we can also use PSIS to approximate the LOO posteriors. A variation of this procedure is implemented in the `loo_R2` method. Is is not exactly the same as this sketched LOO-CV approach for R^2_{basic}), but conceptually quite similar. Let's check what `loo_R2` thinks of our Gaussian and Student-t models:

```
loo_R2(fit_epi_gaussian3)
```

```
    Estimate Est.Error      Q2.5     Q97.5
R2 0.5675538 0.0926324 0.3618969 0.7216547
```

```
loo_R2(fit_epi_student1)
```

```
    Estimate  Est.Error      Q2.5     Q97.5
R2 0.3786142 0.04279025 0.3017956 0.4686028
```

Interestingly, the Student-t model seems to be doing worse here, similar to what we had seen for some of the RMSE metrics. The same pattern can be observed when running `bayes_R2` (try it out as an exercise), so the reason for this behavior does not lie in the fact that we look at out-of-sample predictions. Since both `loo_R2` and `bayes_R2` use `posterior_epred` to create predictions, the Student-t model does not suffer from its abysmal predictions that we may sometimes obtain when running `posterior_predict` with this likelihood. Accordingly, it is not immediately intuitive why the results look like this. I must admit I was a bit puzzled myself when I first saw such results. But it does make sense if you think about it in the following way: Every R^2 contains an estimate of the aleatoric uncertainty in its numerator in some form of error variance. The higher this estimate of aleatoric uncertainty, the lower (worse) the estimate of R^2. If a model with an insufficiently flexible likelihood, such as our Gaussian model, is faced with non-conforming observations ("outliers" from the perspective of the model), it will shift its mean in their direction rather than (primarily) increasing its estimate of aleatoric uncertainty. As a result, it will yield greater (better) R^2 values than an alternative model with a more flexible likelihood, such as our Student-t model, which has an easier time excepting non-conforming observations as aleatoric, therefore increasing its estimate of aleatoric uncertainty. It is unfortunate that a model being willing to accept greater aleatoric uncertainty is punished for it by R^2 metrics. For this reason, I would advice against comparing models with different understandings of aleatoric uncertainty using R^2 metrics unless one has an explicit rational for doing so. In particular, this concerns models having different likelihood families, but it may also concern models having the same likelihood family, but different modeling of the parameters managing the aleatoric uncertainty. We will discuss the latter kind of model in Chapter DISTRIBUTIONAL.

2.8 Prior predictive performance

So far, we have been focusing on measures of posterior predictive performance. That is, we have first fitted the model and then evaluated it's prediction based on the posterior distribution of parameters after seeing the (training) data. We will now take a fundamentally different approach by evaluating *prior predictive distribution*, that is, by looking at what predictions a model implies before seeing any data. As if our training data were empty and all data were belonging to the test set.

For posterior predictive performance, the prior mattered only insofar it affected the posterior, which is often not so much, if the model is relatively simple compared to the amount of data. For prior predictive performance, the prior will always matter strongly, no matter how much data we have. So we have to be much more thoughtful when specifying our priors. In particular we should avoid improper or very vague priors for reasons I will explain soon. We this warning in mind, let's begin.

2.8.1 Prior predictive checks

The starting point to investigating prior predictive performance is to perform graphical prior predictive checks. In brms, they can be obtained in an almost identical fashion than posterior predictive checks. All we have to do is to tell brms that it should not consider the data when sampling from the posterior. What remains is, of course, only the prior. For this to work we need to make sure that all parameters have proper priors though, and ideally not any too ridiculous priors either. Let's go ahead and explicitly specify some proper, weakly informative priors for our basic Gaussian interaction model on the epilepsy data:

```
prior_epi_gaussian6 <-
  prior(normal(6, 3), class = "Intercept") +
  prior(normal(0, 5), class = "b", coef = "Trt1") +
  prior(normal(0, 1), class = "b", coef = "Base") +
  prior(normal(0, 1), class = "b", coef = "Trt1:Base") +
  prior(normal(0, 15), class = "sigma")
```

In the `normal(5, 3)` prior for `Intercept` we are encoding that a priori we are expecting a mean count of 6 but that it may very well be also 3 or 9; even values of 0 or 12 would be quite possible. In fact around 95% of the prior probability lies between 0 and 12, following the mean ± 2 SD rule of the 95% central uncertainty interval of normal distributions. Remember that the priors we put on `class = "Intercept"` are not in fact directly targeting the actual intercept when all predictors are taking on the value of zero, but rather an alternative intercept when all predictors are at their mean. This primarily has computational reasons but, as a side effect, also makes prior specification of the intercept easier in brms models.

In the `normal(0, 5)` of the treatment main effect, we are encoding that we assuming a treatment effect in the interval $[-10, 10]$ with 95% probability, for those patients with a baseline seizure count of 0. Remember that the main effect of a predictor can only be interpreted in the context of corresponding interactions involving said predictor, which doesn't make prior specification any easier. In our case, only very few participants have a baseline seizure count of 0, which makes the main treatment effect measure an almost hypothetical scenario and highlights the difficulties in prior specification even in such very simple regression models. The specification of the other priors on regression coefficients follows a similar logic, which I invite you think through as an exercise.

The `normal(0, 15)` prior on the residual standard deviation requires a bit more elaboration perhaps. As we remember from earlier, Stan automatically truncates prior at the parameters boundaries. Accordingly, we in fact have specified a *truncated* normal prior with lower truncation bound of zero, since standard deviations cannot become negative. If you want to imagine this prior, just take a normal prior with mean 0 and SD of 15 and then cut off the lower half of it below zero. For this reason a truncated normal prior with a mean of zero before truncation is also referred to as a *half*-normal prior. The SD of 15 encodes that we assume a

lot of random noise in our data that we cannot explain by means of our predictors. In fact, we high probability we do expect σ values from very close to zero up to $2 * \text{SD} = 30$ seizure counts.

You see, in order to even begin to sensibly evaluate prior predictive performance, we need to think really hard about priors. Now that we are done, we are "fitting" the model with the option `sample_prior = "only"`, which ensures that Stan ignores the likelihood contribution to the posterior, such that the posterior directly resembles the prior. To be fair, we could also sample from the prior directly by sampling from the bunch of normal specified above. However, we then had to manually create a brms model around those prior draws, which is not the most convenient thing to do. What is more, some priors are actually hard to sample from directly, so brms comes with the `sample_prior = "only"` to cover all these cases and make the prior predictive checking workflow easier for you to code. We also still need to pass the dataset to the `data` argument such that brms knows about the meaning of all variables in the formula. Otherwise, brms would, for example, not know which variables would be factors and what levels the factors would have.

```
fit_prior_epi_gaussian6 <- brm(
  count ~ 1 + Trt * Base,
  data = epilepsy,
  prior = prior_epi_gaussian6,
  sample_prior = "only"
)
```

When looking at the model summary, we see that the estimated "posteriors" indeed seem to resemble the priors:

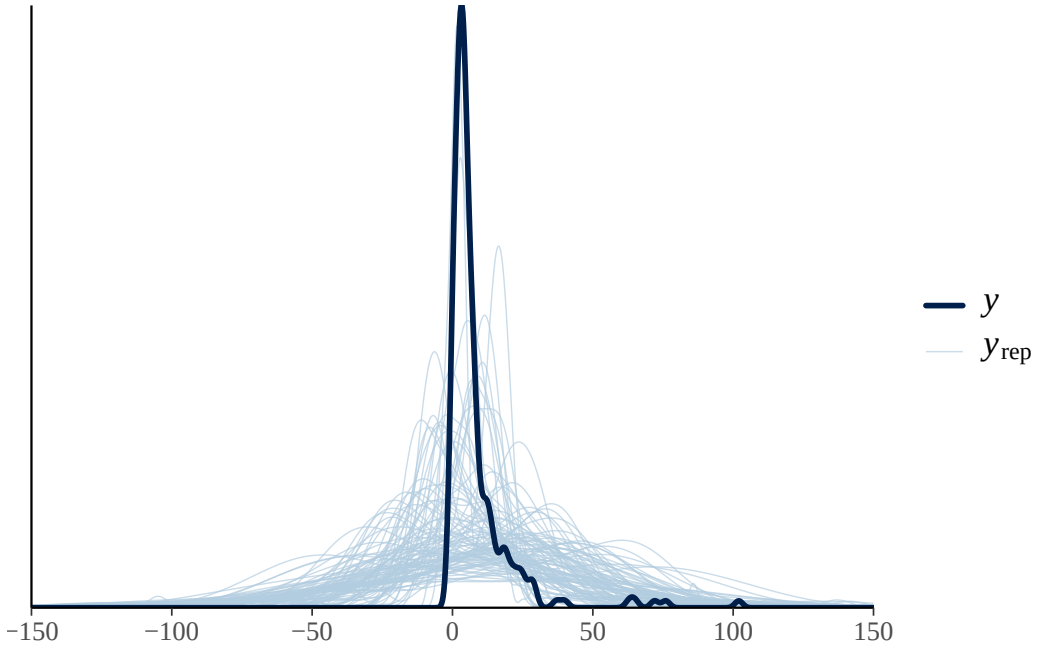
```
summary(fit_prior_epi_gaussian6)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: count ~ 1 + Trt * Base
   Data: epilepsy (Number of observations: 236)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     6.20     35.75   -64.79    73.80 1.00     3962     2851
Trt1          0.05      4.99    -9.40     9.85 1.00     4055     2742
Base         -0.01      1.01    -1.94     1.98 1.00     3794     2854
Trt1:Base    -0.00      0.97    -1.91     1.88 1.00     4206     2814
```

```
Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    11.96      9.14     0.50    34.46 1.00     2678     1613

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

For all practical purposes, our "prior-only" brms model can be post-processed as any other brms model. In particular, we can call `pp_check` to obtain prior predictive checks:

```
pp_check(fit_prior_epi_gaussian6, ndraws = 100) + xlim(-150, 150)
```



We restrict the x-axis a bit to avoid occasional outliers that may cause the plot to be unreadable. When doing so, please make sure that you are not accidentally loosing most of your draws to the axis truncation, which would result in a misleading impression of our prior predictive distribution. The marginal prior predictive check above overall looks reasonable. The prior predictive distribution is wider than the observed data distribution, but roughly on the same order of magnitude. That's what we would hope to see, at least marginally. It doesn't ensure with certainty that the priors are all reasonable but at least non is ridiculously wide. Of course our Gaussian model is still a bit stupid in that it predictive many negative counts. Since its likelihood doesn't have a lower bound at zero, that is not a lot we can do that would not involve

making our prior very (perhaps overly) narrow. So we consider this model's prior predictive distribution as "good enough" for our purposes here.

Earlier, we have spend considerable amount of time to compare the Gaussian with the Student-t interaction model. We will continue this theme here. So let's do some prior predictive checks for the Student-t model too. Compared to the Gaussian model, we only have one more parameter, the degrees of freedom ν (`nu`). By default, ν already has a reasonable, weakly informative `gamma(2, 0.1)` prior, which we specify below manually just to make it explicit:

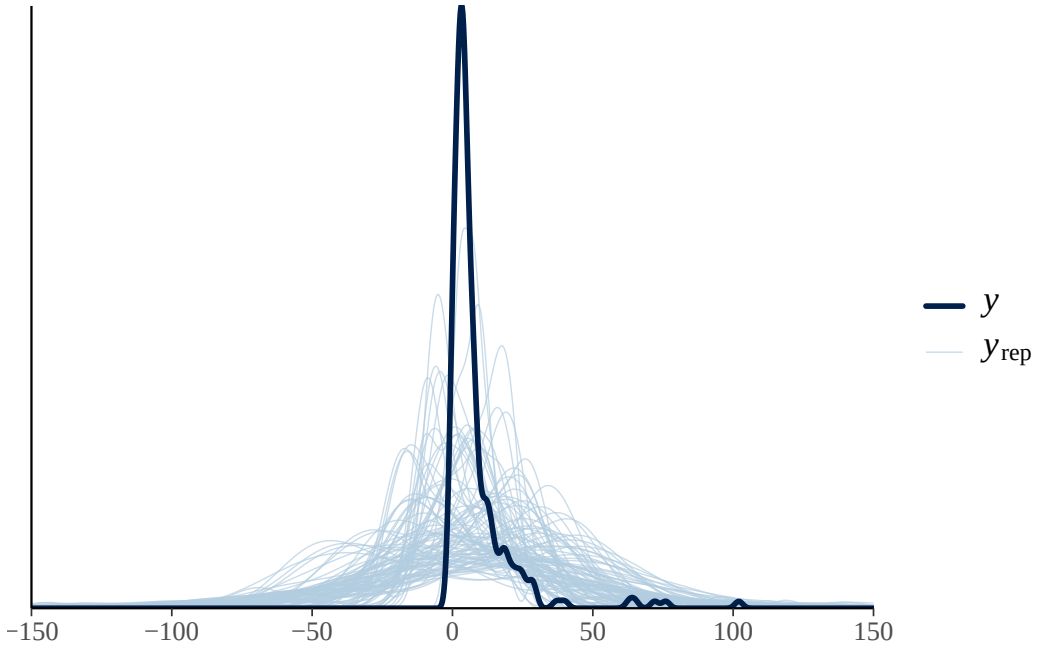
```
prior_epi_student6 <- prior_epi_gaussian6 +
  prior(gamma(2, 0.1), class = "nu")
```

All other parameters are shared between the Gaussian and the Student-t model. The σ parameter slightly changes its meaning as it is no longer exactly the residual standard deviation in Student-t models, but since it still conveys roughly the same thing, we keep its prior untouched for our purposes here, hoping that the prior is still reasonable enough.

```
fit_prior_epi_student6 <- brm(
  count ~ 1 + Trt * Base,
  data = epilepsy, family = student(),
  prior = prior_epi_student6,
  sample_prior = "only"
)
```

After "fitting" the prior-only model, we can again perform prior predictive checks:

```
pp_check(fit_prior_epi_student6, ndraws = 100) + xlim(-150, 150)
```



Doesn't look too bad and actually relatively similar to the check of the Gaussian model. If you run this code yourself and pay attention to the warnings, you will see a few more points being excluded from the plot because of the x-axis limits, a result of the fatter tails of the Student-t distribution, which we have discussed extensively above. But nothing too concerning, so it seems we have found some reasonable enough priors for our models to continue our prior predictive endeavors.

2.8.2 Marginal likelihood-based metrics

How are we going to mathematically formalize the prior predictive performance of a model? Earlier in this chapter, when discussing ELPD, we have evaluated the pointwise likelihood in expectation (averaged) over the posterior. To evaluate prior predictive performance, we could do the same thing but average over the prior. And in fact, that is almost what is done in practice, except that we do not evalute the pointwise likelikelhoods $p(y_n \mid \theta)$ individually, but rather the whole (joint) likelihood $p(y \mid \theta)$ directly:

$$p(y) = \int p(y \mid \theta)\, p(\theta)\, d\theta$$

You might recognize this quantity as the marginal likelihood that we find in the denominator of Bayes theorem. And indeed it is exactly this quantity. Another name for the marginal likelihood is "evidence", which is more common in some fields using the marginal likelihood for model comparison. Let's add the model indicator M to the equation, to clarify that we are

referring the marginal likelihood of model M, evaluted from the prior and likelihood of said model:

$$p(y \mid M) = \int p(y \mid \theta, M)\, p(\theta \mid M)\, d\theta$$

Written this way, we can interprete the marginal likelihood as the likelihood of the data given the model. So instead of doing inference about model parameters based on the (original) likelihood, we can do inference about model themselves based on the marginal likelihood. And just as with the absolute likelihood values $p(y \mid \theta)$, the absolute marginal likelihood values $p(y \mid M)$ are very hard to interprete. We only know that higher is better. Accordingly, we have to compare the marginal likelihoods of two (or more) models to make sense of how good or bad a model fits to the data. The most common such comparative metric is the Bayes factors that is defined as the ratio of two models' marginal likelihoods:

$$\text{BF}_{12} = \frac{p(y \mid M_1)}{p(y \mid M_2)}$$

We use BF_{12} to indicate that model M_1 is in the numerator and model M_2 is the the denominator. If the Bayes factor is greater than 1, the data y have a higher likelihood given model M_1 compared to M_2, and vice versa. Admittedly, this interpreation remains a bit awkward. Ideally, we would like to say something like: "Given the data y, model M_1 is more likely than model M_2." And indeed we can make such a statement by considering the posterior odds:

$$\frac{p(M_1 \mid y)}{p(M_2 \mid y)} = \frac{p(y \mid M_1)}{p(y \mid M_2)} \frac{p(M_1)}{p(M_2)} = \text{BF}_{12} \frac{p(M_1)}{p(M_2)}$$

In words, the posterior odds is defined as the ratio of the posterior probabilities of model M_1 vs. M_2 given data y. It can be computed as the Bayes factor multiplied by the prior odds, which indicates the ratio of prior probabilities we assign to M_1 vs. M_2. For example, if we a priori (before seeing the data) think that M_1 is more plausible than M_2, we can set the prior odds to values larger than 1. If we think that a priori both models are equally likely, we simply set the prior odds to 1, in which case the posterior odds are equal to the Bayes factor. In practice, we often see Bayes factors being interpreted as if they were posterior odds, but keep in mind this only works in the special case of the prior odds being 1. Let me add that often we set the prior odds to 1 not actually because we really believe in models being equally likely a priori, but simply out of convenience; just as we often set wide or even completely flat priors on parameters.

Setting aside the challenges of prior specification, the mathematical theory of Bayes factors and posterior odds is pretty straight forward. However, since the Bayes factor is based on marginal likelihoods, the computational challenges are substantial. Fortunately, there is one class of algorithms that enables reliable computation of (log) marginal likelihood on the basis of posterior draws. This class of algorithms is called bridge sampling (Meng and Schilling 2002).

When using bridge sampling, we do not compute the marginal likelihoods directly, e.g., based on prior draws. Rather, we have to obtain draws from the posterior first (e.g., via MCMC) first, which combined with corresponding log unnormalized posterior density values, enables us to obtain a reliable marginal likelihood estimate. Stan provides us with both, posterior draws and log unnormalized posterior densities, so Stan and bridge sampling play well together. The algorithmic details of bridge sampling are not relevant here, but if you want to learn more, I recommend that you check out the tutorial by Gronau et al. (2017). In R, bridge sampling is implemented in the *bridgesampling* package (Gronau, Singmann, and Wagenmakers 2020), a much more descriptive name than, say, brms I must admit.

When wanting to run bridgesampling in brms, we have to keep in mind two special things: (1) Marginal likelihood estimation via bridgesampling usually requires several times more posterior draws than the estimation of posterior moments or quantiles (i.e., what we usually do with posterior draws). Exactly how many posterior draws is sufficient is hard to tell but in my experience something between 3 to 5 times should be good enough, but please don't take my word for it. The bridgesampling package also comes with good diagnostics to inform us if its estimates' reliably is likely insufficient. (2) Bridgesampling requires posterior draws of all variables declared in the parameters block of Stan. This is more of technical requirement. It only concerns brms users because brms does not necessarily store all these variables, since some of them are irrelevant for the usual post-processing. Except for bridgesampling that is. To ensure all these variables are stored, you have to set `save_pars = save_pars(all = TRUE)` in `brm`. If you fail to do so, and you are fitting anything but a very simple model, chances are some parameter variables were not stored and bridge sampling will fail accordingly. In response, you then need to fit your model again with the above `save_pars` option. A bit annoying, I know, but the alternative would be to always store all kinds of variables in a brms model, thus blowing up model size drastically – only to seamlessly support the (within the brms community) relative niche feature of marginal likelihood estimation.

Anyway, here is the code to fit our Gaussian interaction model with the above discussed priors and all setting enabled to support a reliable marginal likelihood estimation:

```
fit_epi_gaussian6 <- brm(
  count ~ 1 + Trt * Base, data = epilepsy,
  prior = prior_epi_gaussian6,
  save_pars = save_pars(all = TRUE),
  iter = 5000, warmup = 1000
)
```

Obtaining an estimate of the log marginal likelihood via bridgesampling is then just a matter of a few seconds.

```
logml_epi_gaussian6 <- bridge_sampler(fit_epi_gaussian6)
```

```r
summary(logml_epi_gaussian6)
```

```
Bridge sampling log marginal likelihood estimate
(method = "normal", repetitions = 1):

 -831.2982

Error Measures:

 Relative Mean-Squared Error: 7.733336e-07
 Coefficient of Variation: 0.0008793939
 Percentage Error: 0%

Note:
All error measures are approximate.
```

The log marginal likelihood estimate of about -831 also clarifies why we are estimating the marginal likelihood on the log scale: If we were to exponentiate this value, computers will just straight up round it down to 0 thus rendering the estimate entirely useless. Of course, without a second model to compare against, our log marginal likelihood estimate, while numerically stable, does not tell us much. After all, marginal likelihoods are only estimates of *relative* prior predictive performance. Let's fit out Student-t interaction model then as well with all bridgesampling settings enabled:

```r
fit_epi_student6 <- brm(
  count ~ 1 + Trt * Base, data = epilepsy,
  family = student(),
  prior = prior_epi_student6,
  save_pars = save_pars(all = TRUE),
  iter = 5000, warmup = 1000
)
```

```r
logml_epi_student6 <- bridge_sampler(fit_epi_student6)
```

```r
summary(logml_epi_student6)
```

```
Bridge sampling log marginal likelihood estimate
(method = "normal", repetitions = 1):
```

```
 -720.6938

Error Measures:

 Relative Mean-Squared Error: 9.616052e-06
 Coefficient of Variation: 0.003100976
 Percentage Error: 0%

Note:
All error measures are approximate.
```

Computing the (log) Bayes factor of the two models is then straightforward:

```
bayes_factor(logml_epi_gaussian6, logml_epi_student6, log = TRUE)
```

```
Estimated log Bayes factor in favor of x1 over x2: -110.60446
```

All what this method does is subtract the two log marginal likelihood estimates since ratios become differences on the log scale. The log Bayes factors estimate of -110 is very negative, which tells us that first model is much worse than the second model. Here again, it pays off to compute things on the log-scale as we will otherwise get a Bayes factor of 0, which would, if we were to take this value literally imply an "infinitely worse" fit of the Gaussian vs. the Student-t model. To be fair, a log Bayes factor of -110 is also very close to "infinitely worse" already.

We can also change the order of the log marginal likelihood estimates, and get the same result only with a reversed sign for the log Bayes factor. This is because ratios become differences on the log scale, so switching numerator and denominator of the Bayes factor is equivalent to the log Bayes factor switching its sign.

```
bayes_factor(logml_epi_student6, logml_epi_gaussian6, log = TRUE)
```

```
Estimated log Bayes factor in favor of x1 over x2: 110.60446
```

If we just want to compute Bayes factors of two brms models, without having to manually call `bridge_sampler`, we can also just use the `bayes_factor` method directly on the two model objects:

```
bayes_factor(fit_epi_gaussian6, fit_epi_student6, log = TRUE)
```

Be aware though that this will recompute the marginal likelihoods via bridge sampling every time we call the method, unless we have previously stored the estimates in the model itself via `add_criterion`:

```
fit_epi_gaussian6 <- add_criterion(fit_epi_gaussian6, "marglik")
fit_epi_student6 <- add_criterion(fit_epi_student6, "marglik")
```

At the start of this section, I emphasized that measures of prior predictive performance will always remain prior dependent regardless of how much data we throw at the model. As a result, when using such measures, we always have to think carefully about our priors. The two interaction models are not ideally suited to demonstrate the point of prior dependence. So let's fit another model, the Gaussian model without interactions from earlier in the chapter, and compare it to the Gaussian interaction model.

```
prior_epi_gaussian7 <-
  prior(normal(6, 3), class = "Intercept") +
  prior(normal(0, 5), class = "b", coef = "Trt1") +
  prior(normal(0, 1), class = "b", coef = "Base") +
  prior(normal(0, 15), class = "sigma")
```

As you see, we will use the same priors as above just that we remove the prior on the interaction term.

```
fit_epi_gaussian7 <- brm(
  count ~ 1 + Trt + Base, data = epilepsy,
  prior = prior_epi_gaussian7,
  save_pars = save_pars(all = TRUE),
  iter = 5000, warmup = 1000
)
```

```
bayes_factor(fit_epi_gaussian6, fit_epi_gaussian7)
```

We obtained a Bayes factor of around 60 in favor of the Gaussian interaction model, which by common conventions would be classified as strong evidence ($BF > 10$). Not that I want you to follow these conventions. I just want to give you a sense on how people might typically interpret this result.

I promised you to showcase prior sensitivity, so let's replace the `normal(0, 1)` prior on the interaction term by a `normal(0, 10)` prior:

```
prior_epi_gaussian6b <- prior_epi_gaussian6
prior_epi_gaussian6b$prior[4] <- "normal(0, 10)"
```

In the above code, we have made use of the fact that `brmsprior` objects are essentially just data.frames and that the prior on the interaction coefficient happend to be in the 4th row as a result of how we defined `prior_epi_gaussian6` initially. Now, let's fit the Gaussian interaction model again with the adjusted prior.

```
fit_epi_gaussian6b <- brm(
  count ~ 1 + Trt * Base, data = epilepsy,
  prior = prior_epi_gaussian6b,
  save_pars = save_pars(all = TRUE),
  iter = 5000, warmup = 1000
)
```

```
bayes_factor(fit_epi_gaussian6b, fit_epi_gaussian7)
```

Apparently, the Bayes factor has reduced substantially and is now around 6. According to common conventions, we would consider this (only) moderate evidence in favor of the interaction model. We are even already nearing the space ($1/3 < BF < 3$), where we would consider the evidence in favor or against any of the models to be inconclusive. Notably, the posterior distribution of the interaction coefficients is virtually unchanged (check the summary output of the models to verify). We see that the Bayes factor may change quite dramatically as a result of changing the prior *even though* the posterior barely changes at all. We can take this even further and change the prior on the interaction coefficient to `normal(0, 100)`, with the result that the Bayes factor shrinks to about 0.6. This would now be in range of inconclusive evidence but with a direction that rather points to the no-interaction model fitting (slightly) better. Again the posterior remains essentially the same. You might argue that both the `normal(0, 10)` and the `normal(0, 100)` prior are ridicolously wide for this interaction coefficient and I agree. My point here is only to showcase the prior sensitivity of Bayes factors (and other marginal likelihood-based metrics), hopefully convincing you to think carefully about your priors if you plan on applying such metrics for model comparison.

After following both the formal and the informal discussions of Bayesians over several years, I came to the conclusions that the dispute of prior vs. posterior predictive performance will never be resolved. And that's alright. Both kind of approaches have their sensible use cases and also their shortcomings. We simply have to know when to apply what. Personally, I am more on the posterior predictive side. Not for any strong theoretical or philosophical reasons, but only because prior predictive metrics are just so terribly inconvenient: Even if I have overwhelming amounts of data, where no reasonable prior will have any relevant influence on the posterior, I would still have to think carefully about my priors; and I just cannot be bothered enough to do this for most analysis.

Let me finish this section by pointing to the rich, but so far practically underused field of (expert) prior elicitation, which enables the translation of expert knowledge into prior distributions (for a recent review see Mikkola et al. 2023). Even after decades of research we are still struggling to bring prior elicitation into the Bayesian analyses of the masses, because it turns out that both the process of eliciting expert knowledge and translating it into priors is actually really hard. It is one of the topics of my research to make this process feasible and user-friendly for a much larger audience (Bockting, Radev, and Bürkner 2023), but we are still quite far away from that goal.

2.9 Model averaging

The results of model comparison can usually be use for two different purposes. The purpose is *model selection* as we have seen above. There, we compare a bunch of models with each other and then select a single one, or a few if we cannot decide for a single one, with whom we will then move forward. The other, non-selected models will then be discarded.

But what if we want to use all models? If we want to use all models "equally" in some sense, for example to evaluate predictive performance, then we can just take posterior predictive draws from all models and pool them together in a single vector of draws, which we then process further as if the draws where all coming from a single model. In this case, we wouldn't need to do model comparison at all. But it also sounds a bit wrong to use all models equally, especially if we know than some models are predicting much better than others. So it is perhaps more sensible to give better predicting models higher weight, and more poorly predicting models less weight. Suppose we have two models, and our model comparison procedure indicated that model M_1 was 3 times better on some metric than model M_2. Translating that into weights w, that is, non-negative values that sum to 1, this would imply the weight of M_1 to be $w_1 = 0.75$ and the weight of M_2 to be $w_2 = 0.25$. If we know wanted to combine the two models posterior predictive distributions, we could select 75% of the draws from M_1 and only 25% of draws from M_2, and then only pool the select draws. The general procedure of combining model's predictions according to some kind of weights is called *model averaging*. Below, I will briefly describe how to obtain weights from marginal likelihood-based and information criterion-based procedures.

2.9.1 Weights from marginal likelihoods

In Section 2.8.2, we have discussed comparing the marginal likelihoods of two models using Bayes factors (or posterior odds), but what shall we do if we have more than two options in our set of models being considered? Of course, we would compute the Bayes factors of each of the pairwise comparisons but that would be quite cumbersome to interprete. To improve the situation, we can compute the the posterior model probabilities $p(M_j \mid y)$ for all considered

models M_j explicitly, under the assumption that the true model is within the set of considered models. This leads us to apply Bayes theorem but to models instead of to parameters:

$$p(M_j \mid y) = \frac{p(y \mid M_j)\, p(M_j)}{\sum_{j'=1}^{J} p(y \mid M_{j'})\, p(M_{j'})}$$

In words, we multiple each model's marginal likelihood with its prior model probability and normalize it with the sum of these products over all considered models. In brms, we can compute posterior model probabilities using the `post_prob` method. Here, we assume equal prior model probabilities for the lack of any more sophisticated idea:

```
post_prob(fit_epi_gaussian6, fit_epi_student6, prior_prob = c(0.5, 0.5))
```

```
fit_epi_gaussian6  fit_epi_student6
      9.23101e-49       1.00000e+00
```

Unsurprisingly, we obtain a posterior model probability of basically 100% for the Student-t model and one of basically 0% for the Gaussian model. That is, if one of these two models was indeed the true model, than we would be almost 100% sure it is the Student-t model.

Since brms supports several different model weighting methods (see below), it provides the `model_weights` method as a unified interface among all of them. If we want to compute posterior model probabilities using the `model_weights` method, we can use the `weights = "bma"` option, where "bma" stands for Bayesian model averaging. This will internally call `post_prob` so is equivalent to the code above.

Theoretically, posterior model probabilities are a very appealing quantity since they literally encode the probability of each considered model given the data. However, it comes with the key assumption that, as already mentioned above, the true model is among the set of considered models – something we also call the *closed-world assumption* (Bernardo and Smith 1994). In other words, we are assuming that we consider all theoretically possible models in our comparison. If you think about it, that is a very strong assumption and rarely justified unless we are working in a field with very strong theories that we can turn into equivalent statistical models. Perhaps this is justified in some parts of physics, but I struggle to see this being the case much more generally.

If we let go of this assumption, and allow the true model to be outside of the scope of considered models – or if we don't even assume the existence of a true model in the first place – we talk about an *open-world assumption* (Bernardo and Smith 1994). Typically, measures of prior predictive performance based on marginal likelihoods are considered in the context of a closed world, while measures of posterior predictive performance such as ELPD are considered in the context of an open world. I am not sure this is really a necessary distinction or rather follows

purely from the history of individual fields that have concerned themselves with Bayesian model comparison.

Clearly, for the epilepsy data, none of the two models considered above is even close to the truth, which demonstrates the key limitation of posterior model probabilities and their closed-world assumption. Accordingly, in most cases, I would not interpret posterior model probabilities as indicating the probability of a given model being true but rather only as a metric that shows relative prior predictive model performance on an intuitive probability scale.

2.9.2 Weights from ELPD scores

In the context of posterior predictions, we have talked a lot about the expected log predictive density (ELPD) as a metric for model comparison. It is surprisingly easy to turn them into model weights as well. The ELPD is essentially the sum of log probabilities $\log p(\tilde{y}_n \mid y, M)$ where $\tilde{y}_n$ are the observations in the test data, y are the training data (see Section 2.7 for details). Accordingly, we can exponentiate it to obtain the product of probabilities $p(\tilde{y}_n \mid y, M)$. When we assume conditional independence, this product is equal to the joint probability $p(\tilde{y} \mid y, M) = p(\tilde{y}_1, \tilde{y}_2, ... \mid y, M)$ over all $\tilde{y}_n$ in the test data given the training data. The higher this joint probability, the better the predictions of the model on test data. This motivates to define ELPD weights simply as the exponentiated ELPD scores normalized against their sum over all considered models:

$$w_j = \frac{\exp(\text{ELPD}_j)}{\sum_{j'=1}^{J} \exp(\text{ELPD}_{j'})}$$

Sometimes, these weights are also called pseudo-BMA weights due to their structural similarities with posterior model probabilities If we compute these weights directly, we may run into numerical issues since the ELPD scores may be so small that they underflow to 0 after exponentiation. To mitigate this issue, we first compute the maximum of the ELPD values and subtract it from all ELPD values, $\Delta\text{ELPD}_j = \text{ELPD}_j - \max(\text{ELPD})$, before exponentiation:

$$w_j = \frac{\exp(\Delta\text{ELPD}_j)}{\sum_{j'=1}^{J} \exp(\Delta\text{ELPD}_{j'})} \tag{2.1}$$

Mathematically, this doesn't change the result, but numerically it makes things much stable. I suggest you to try it out yourself as an exercise. And by the way, we also use this trick to make the computation of posterior model probabilities more numerically stable. I just didn't mention it above to focus on other aspects.

In brms, we can compute model weights based on LOO-CV's ELPD values as follows:

```
model_weights(fit_epi_gaussian6, fit_epi_student6, weights = "loo")
```

```
fit_epi_gaussian6  fit_epi_student6
     1.170659e-55      1.000000e+00
```

Also here, we see very clear evidence that the Student-t model outperforms the Gaussian model. For this case here, marginal likelihood and ELPD-based model weights are in agreement, but this is not generally the case. Depending on the model, data, and priors, there might as well point in opposite directions.

The model weighting approach presented here does not work only for ELPD values but actually for all methods that can produce values on the "information criterion scale". Remember that, if we multiply the ELPD with -2, we get an information criterion, which we simply called LOOIC for ELPDs computed from LOO-CV. If we rewrite Equation 2.1 in terms of information criteria, we get what is known in the literature as *Akaike weights*. If you want to learn more, check out Wagenmakers and Farrell (2004) for a relatively gentle introduction.

2.9.3 Weights from stacking of predictive distributions

Lastly, I want to briefly discuss another option for model weighting, which is called *stacking of posterior predictive distributions* (Yao et al. 2018). In a nutshall, we are trying to find the model weights $w = (w_1, \dots, w_J)$ for our models $(M_1, \dots, M_J)$ such that the test data have maximum probability under the weighted average of the models' posterior predictive distributions. In math, we can write this as follows:

$$\max_{(w_1,\dots,w_J)} \frac{1}{\tilde{N}} \sum_{n=1}^{\tilde{N}} \log \sum_{j=1}^{J} w_j \, p(\tilde{y}_n \mid y, M_j)$$

This general approach can be readily combined with all kinds of cross-validation schemes. For example, when combined with LOO-CV, we set $p(y_n \mid y_{-n}, M_j)$ as our posterior predictive distribution $p(\tilde{y}_n \mid y, M_j)$. As a result, we can also efficiently approximate LOO-CV based stacking with importance sampling, just as we do for LOO-CV-based ELPD scores.

In order to obtain stacking weights with brms, we can use

```
(ws <- model_weights(fit_epi_gaussian6, fit_epi_student6, weights = "stacking"))
```

```
Warning: Some Pareto k diagnostic values are too high. See help('pareto-k-diagnostic') for de
```

```
fit_epi_gaussian6  fit_epi_student6
        0.1383116         0.8616884
```

For our two example models, stacking leads to somewhat less extreme weights than the other weighting methods we have tried out before, although the Student-t model is still clearly favored. More generally, compared to ELPD (pseudo-BMA) weights, stacking weights seem to perform better in particular when there are many models considered at the same time (Yao et al. 2018). The drawback of stacking weights is mainly the computational overhead created by the optimization step. However, in my experience, this is not too big of a factor and should rarely limit practical applicability. If you want to compute some kind of model weights but have no clear idea or reasoning which specific method to use, I recommending using stacking weights. For this reason, `weights = "stacking"` is also the default in the `model_weights` method.

At the start of this section, I told you that we can use model weights among others to weight posterior predictive distributions. And that's what we are going to now. brms comes with the `pp_average` method which takes a bunch of models as well as weights as input and outputs a weighted posterior predictive distribution:

```
ppa <- pp_average(fit_epi_gaussian6, fit_epi_student6, weights = ws,
                  ndraws = 4000, summary = FALSE)
```

```
str(ppa)
```

```
 num [1:4000, 1:236] -1.001 0.995 2.423 6.16 14.343 ...
 - attr(*, "weights")= Named num [1:2] 0.138 0.862
  ..- attr(*, "names")= chr [1:2] "fit_epi_gaussian6" "fit_epi_student6"
 - attr(*, "ndraws")= Named num [1:2] 553 3447
  ..- attr(*, "names")= chr [1:2] "fit_epi_gaussian6" "fit_epi_student6"
```

Through setting `summary = FALSE`, we got an output that is structurally identical to that of `posterior_epred`, just that parts of the posterior draws (around 14%) have been drawn from the Gaussian model and the remaining draws (around 86%) have been drawn from the Student-t model. In the `weights` attribute, we can even see exactly how many draws were used from which model. In the `weights`, we passed the pre-computed stacking weights directly, but we could have also just named a method like `"stacking"`, `"bma"`, or `"loo"` to compute weights on the fly using the `model_weights` method. With `pp_average`, we can not only average posterior predictive distributions, but also, for example, means of posterior predictive distributions (`posterior_epred`) or linear predictors (`posterior_linpred`). Exactly which predictive quantity is to be averaged can be controlled via the `method` argument.

2.10 In-distribution vs. out-of-distribution

Above, we have learned about the difference between in-sample and out-of-sample data. The former was the data we fit our model on. The latter was the data we used to evaluate predictive performance of the model in an effort to (ideally) avoid overfitting. Now, we will go one step further and split out-of-sample data into *in-distribution* and *out-of-distribution* data. This is not necessarily a brms topic but understanding it was quite illuminating to me so I want to share it with you. In a nutshell, if new data were generated by the same true distribution as the training data, we call the new data "in-distribution". In this case, can have justified hopes that our model, if built and trained properly, will be able to predict these new data well. Conversely, if the data were generated from a *different* true distribution than the training data, we call the new data "out-out-distribution".

Why does this matter? Let me elaborate with a bit more math. Suppose, in reality, the training (in-sample) data $\tilde{y}$ has been generated by a true data generating distribution $p^*(y)$, i.e., $\tilde{y} \sim p^*(y)$. This true distribution $p^*(y)$, will almost never be fully known to us, but it suffices to assume $p^*(y)$ exists. Suppose now we have a new dataset y^* that we didn't use for model fitting. Then, we call y^* "out-of-sample" data (in a strict sense) if it was generated from the same true distribution as the training data, that is, if also $y^* \sim p^*(y)$. Now suppose that our model $p(y \mid \theta)p(\theta)$ is able to learn a good representation of $p^*(y)$ from $\tilde{y}$, that is, if the posterior predictive distribution of our model, $p(y \mid \tilde{y})$ is a good approximation of $p^*(y)$. Then, our model should be able to predict y^* well even though our model hasn't seen these data during training. This is because the same underlying patterns apply for both $\tilde{y}$ and y^*.

However, sometimes the world changes between gathering our training data and the time we use our model for prediction of new y^*; or we gathered y^* at a different spatial location, different population of individuals, etc. In any of these cases, y^* may no longer be truly generated from the original $p^*(y)$ but from some other distribution $p^*_{\text{new}}(y)$. This kind of new data, originating from a different data-generating process than the training data, we call "out-of-distribution" data. Accordingly, if $p^*_{\text{new}}(y)$ differs from $p^*(y)$ in a relevant manner, then, no matter how accurate our model was to approximate $p^*(y)$, it may still fail terribly in predicting $y^* \sim p^*_{\text{new}}(y)$.

Admittedly, in reality, it is hard to test whether new data is just out-of-sample or actually out-of-distribution, because we don't have direct access to $p^*(y)$ and even less so to $p^*_{\text{new}}(y)$. There are whole fields of research dealing with questions arising in this context. You can find the related literature, for example, under the terms "out-of-distribution detection" and "anomaly detection".

The concepts of in- and out-of-distribution data cannot just be applied to true data generating distributions but also more directly to our own Bayesian model $p(y \mid \theta)p(\theta)$. There, we can say that y^* is "in-distribution" (from the perspective of our model), if it is plausible that y^* was generated by the posterior predictive distribution of the model, that is, if it is plausible that $y^* \sim p(y|\tilde{y})$. Conversely, if the assumption of $y^* \sim p(y|\tilde{y})$ is *implausible*, than we could call y^*

to be "out-of-distribution" (from the perspective of our model). If used in the context of model comparison via PSIS LOO-CV, we have seen that the Pareto k diagnostic will tell us something about how influential individual observations are. When our data is not super small, every single data point should only have very little influence. At least if the model is a plausible data generating process, that is, if the data is "in-distribution" for our model. Accordingly, if the Pareto k diagnostic finds influential observations, those are likely "out-of-distribution" for our model. Hence, our model may not be a good approximation for $p^*(y)$ after all. In other words, we can use the distribution of the Pareto k values as an *absolute* fit metric usable to provide evidence against a given model. That said, we should not use the Pareto k values as evidence in favor of a given model, since even bad models may find all observations as "in-distribution" often enough.

Let's illustrate this idea on a concrete example by investigating the Pareto k values for the Gaussian and Student-t models fitted above, more specifically `fit_epi_gaussian3` and `fit_epi_student1` both modeling an interaction between treatment and baseline seizure count:

```
khats_epi_gaussian3 <- loo_epi_gaussian3$diagnostics$pareto_k
khats_epi_student1 <- loo_epi_student1$diagnostics$pareto_k
```

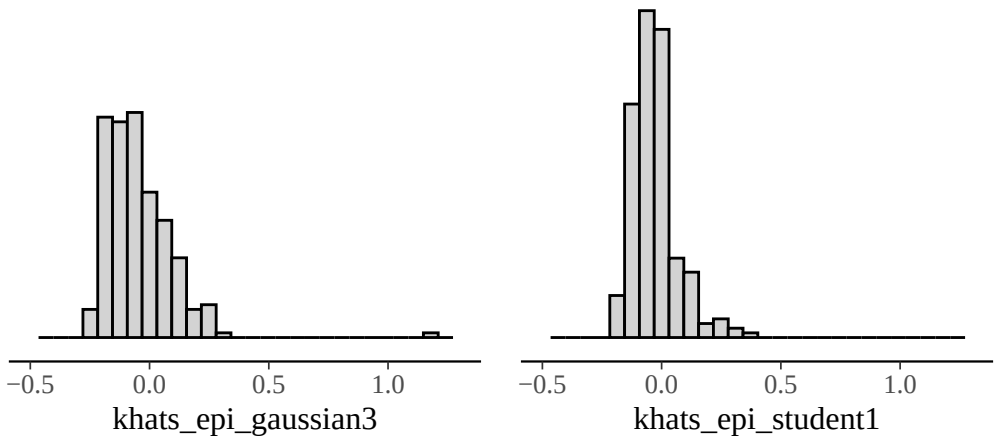


Figure 2.7: Histograms of the LOO Pareto k estimates obtaind for the Gaussian and Student-t interaction models.

As we can see from Figure 2.7, the range of estimate Pareto k estimates is pretty similar for both models, except for two large outliers in the normal model. This indicates that the two corresponding observations are out-of-distribution for the normal model but not for the Student-t model, suggesting that at least the normal model is not such a great fit for the given data, at least if we trust that the two outlying observations are valid themselves. Of course, the fact that the Pareto k estimates are all good for the Student-t model doesn't necessarily make it a good model, but at least no observation lies in the out-of-distributions space.

2.11 Summary

In this chapter, we have learned about the basics, and some more advanced aspects, of Bayesian model comparison. We have seen that this is quite a challenging topic with many caveats and things to keep in mind. Knowing about this complexity should of course not discourage you from doing model comparison but rather help you understanding the implications and limitations of the decisions you make. In most cases, there is no single "right" way of comparing models. There are simply too many degrees of freedom in the Bayesian workflow (Gelman et al. 2020; Bürkner, Scholz, and Radev 2023a). Attempting to find that single "right" way is thus futile. Rather, I suggest you to aim for what you believe are "good enough" methods, taking into account both the analysis goals and practical constraints you face. For example, LOO-CV is not always the most appropriate cross-validation scheme, but if you have to choose between performing PSIS LOO-CV and no cross-validation at all (e.g., because the computational demands of K-fold CV are too high), then it may very well be sensible to opt for PSIS LOO-CV still – provided that its diagnostics indicate sufficiently trustworthy results. Please just make sure to communicate your model comparison methods' properties, implications, and limitations clearly.

3 Generalized Linear Models

3.1 Setup

```
library(dplyr)
library(magrittr)
library(ggplot2)
library(patchwork)
library(bayesplot)
library(brms)
library(knitr)
```

3.2 Introduction

Generalized linear models (GLMs) are very similar to the linear models we have already seen, except that we are choosing a different likelihood distribution other than normal, … and then deal with the consequences that follow from it. Suppose our responses y can only take on positive ($y > 0$) or non-negative ($y \geq= 0$) values, as in the `epilepsy` data where our response is the number of epileptic seizures patient have during a certain time interval. Clearly the number of seizures cannot be negative. If we have only one aspect of the likelihood that we can predict, we will probably want to choose a measure of central tendency as our target, usually the mean or the median. Now, in the described case, the mean or the median will of course also be positive (or at least non-negative), but our linear predictor η doesn't know about this restriction if we don't help it a little bit. That is, we either have to transform our response y to be unbounded and thus on the same scale as η or we have to transform η to be bounded (e.g., positive only) to be on the same scale as y. Specifically the latter case (transforming η) is what leads to GLMs in a more strict sense (McCullagh 2019). But for the purpose of this chapter and the book more generally, I use the term "GLM" more generally as a model with a linear predictor where the likelihood is non-normal and/or we have to transform either y or η for our predictions to stay within the range of possible response values.

3.3 GLMs for lower-bounded responses

To illustrate the use of GLMs for lower-bounded responses, we will continue using the `epilepsy` data. Over the course of this book, we will see lots of other GLMs or their extensions modeling lower-bounded responses. Sometimes the responses are counts, as in the `epilepsy` case, sometimes there are (discrete or continues) times, such as in time-to-event or response time data, or sometimes there are other things that has a lower bound by nature or design. Most of the time, this lower bound will be 0. But even if that bound was some $c \neq 0$, we could transform our responses as $y - c$, such that the new lower bound would be 0 again. As a result, we will focus on this special case knowing that other bounds can be easily expressed as that special case.

While we discuss count responses here, we will see that most of the discussed challenges and solutions apply more generally to other kinds of lower-bounded responses too.

3.3.1 Modeling log counts

Before we start using likelihoods different than Gaussian, let's first stay in the Gaussian regime a bit longer. As we have seen, the Gaussian likelihood leads to predictions of negative (i.e., impossible) counts, which is definitely something we want to avoid. One way to solving this issue is to transform the response variable such that the assumption of a Gaussian likelihood is more reasonable. One transformation that is commonly applied in for positive data is the log transformation. However, for our count response, the log transform is inappropriate since count of zero lead to $\log(0) = -\infty$, which clearly is not very useful for further analysis. But we are not giving up so easily. How about just adding 1 to the counts before log transforming, that is, apply $\log(y + 1)$? That works since $\log(0 + 1) = \log(1) = 0$. This is kind of a hack but actually quite commonly used in some fields (CITE). It's not my preferred approach but it's a good starting point still:

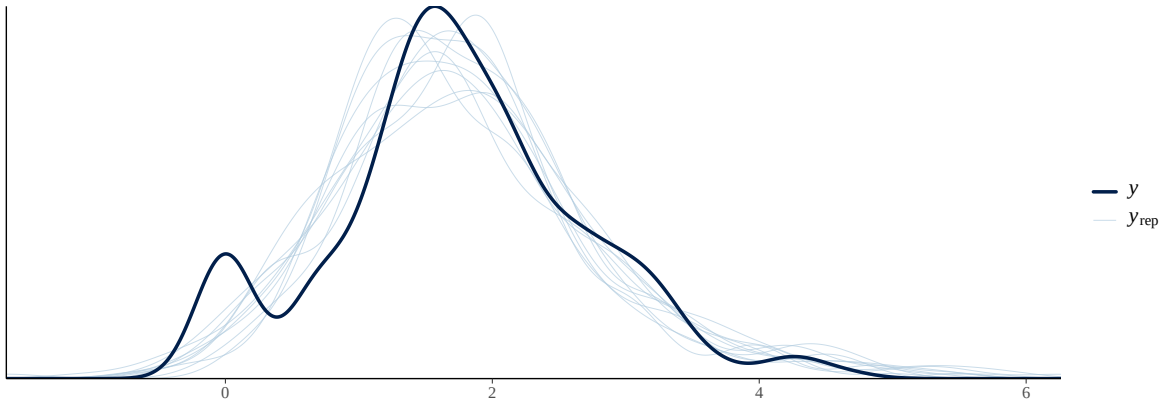
```
fit_epi_gaussian_log1 <- brm(log(count + 1) ~ Trt * scale(Base),
                             data = epilepsy)
```

```
summary(fit_epi_gaussian_log1)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: log(count + 1) ~ Trt * scale(Base)
   Data: epilepsy (Number of observations: 236)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000
```

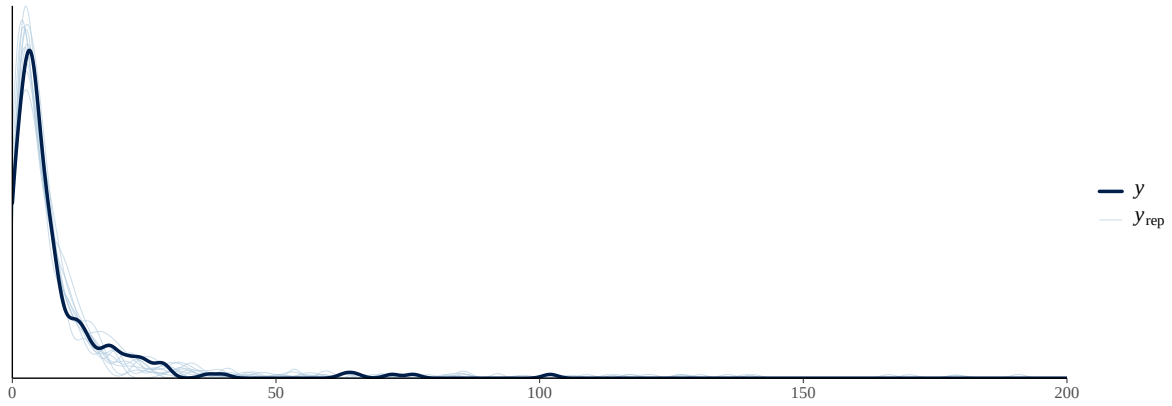
```
Regression Coefficients:
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept         1.87      0.07     1.74     1.99 1.00     4240     3027
Trt1             -0.23      0.09    -0.40    -0.05 1.00     4126     2548
scaleBase         0.63      0.07     0.49     0.76 1.00     2678     2463
Trt1:scaleBase    0.04      0.09    -0.14     0.22 1.00     2692     2427

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.70      0.03     0.64     0.76 1.00     5127     2990

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

```r
pp_check(fit_epi_gaussian_log1)
```



Notice that the response scale is now $\log(y+1)$ so we don't see the original scale in the plots anymore because for brms $\log(y+1)$ is now the actual response variable. If you want to see predictions on the original scale, we have to apply the inverse transformation ourselves. If $z = \log(y+1)$, then $y = \exp(z) - 1$. Accordingly:

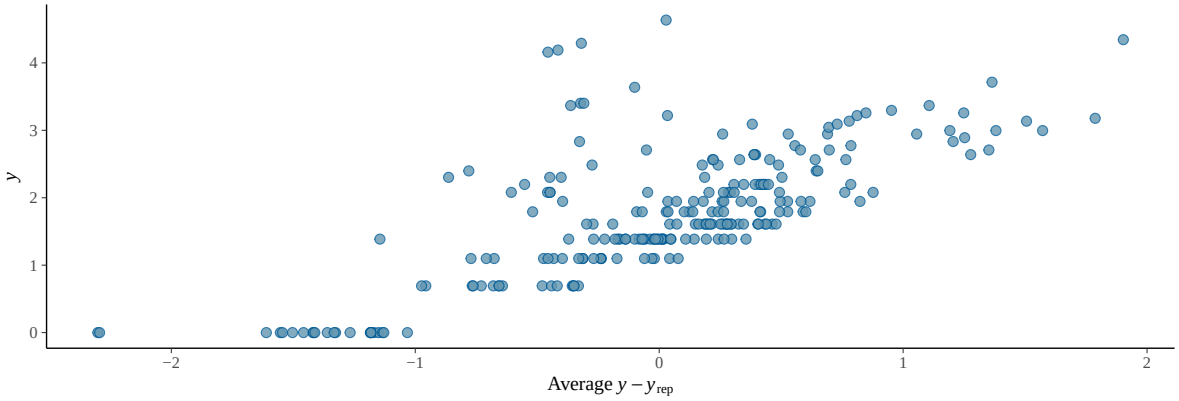
```r
ypred <- posterior_predict(fit_epi_gaussian_log1, ndraws = 10)
ypred <- exp(ypred) - 1
pp_check(epilepsy$count, yrep = ypred, fun = "dens_overlay") +
  xlim(c(0, 200))
```



Doesn't look too bad actually. Notice that, in the above code, we have used the default `pp_check` method that requires us to manually enter the observed responses as the first argument and the predicted responses as the second argument `yrep`.

But what about the residuals? Let's plots them at the log-scale:

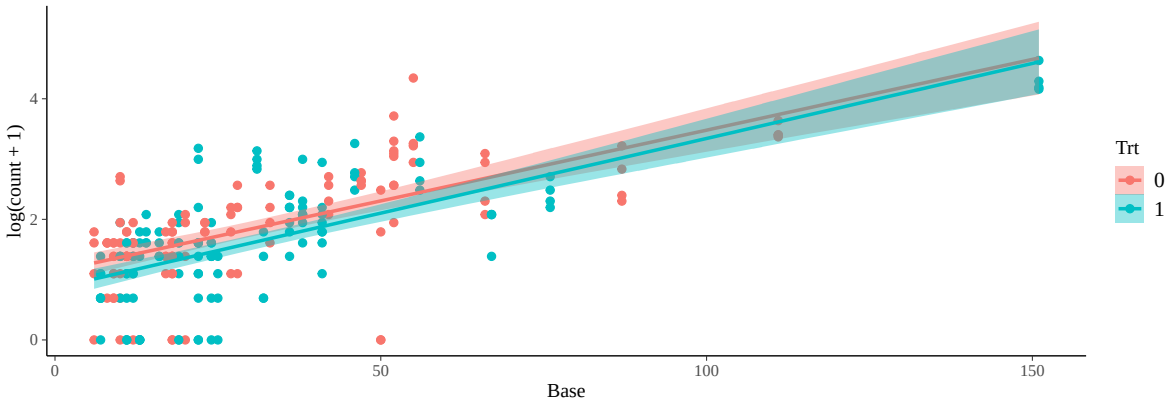
```
pp_check(fit_epi_gaussian_log1, type = "error_scatter_avg")
```



The residual plot shows a clear relationship between residuals and responses, indicating a lot of room for improving our model. We would see the same qualitative picture if we plotted residuals on the original scale, only that outliers would be more clearly visible.

Setting aside our model criticism for now, what does the model think about the effects of seizure baseline and treatment?

```
conditional_effects(fit_epi_gaussian_log1, effects = "Base:Trt")
```



Again, we are seeing conditional predictions on the log-scale where they are linear by choice of using a linear model. Let's build a basic version of `conditional_effects` ourselves to (1) produce conditional predictions on the original scale and (2) illustrate more of the internal workflow that happens inside brms.

First, we set up a dataset containing the predictor values as which we want to evaluate predictions:

```
newdata <- epilepsy %$%
  expand.grid(
    Base = seq(min(Base), max(Base), length.out = 100),
    Trt = unique(Trt)
  )
```

The `%$%` of the `magrittr` essentially allows us to use the objects stored on the left-hand side (i.e, the `epilepsy` dataset) in the right-hand expressions, as if we had written `epilepsy$<variable>` everywhere. It's base R equivalent is the `with` function.

Second, call `posterior_epred` on the new data to obtain draws from the mean of the posterior predictive distribution:

```
pred <- posterior_epred(fit_epi_gaussian_log1, newdata = newdata)
str(pred)
```

```
 num [1:4000, 1:200] 1.27 1.27 1.37 1.37 1.1 ...
```

Third, transform and subsequently summarize posterior predictive mean samples. As a general rule, *always* (if possible) transform first on the level of the posterior draws and summarize only afterwards.

```
pred_exp <- pred %>%
  { exp(.) - 1 } %>%
  posterior_summary() %>%
  bind_cols(newdata)

head(pred_exp, 3)
```

```
  Estimate Est.Error     Q2.5    Q97.5     Base Trt
1 2.596988 0.3298574 1.996278 3.271859 6.000000   0
2 2.721639 0.3318745 2.109655 3.396354 7.464646   0
3 2.850663 0.3339710 2.234789 3.539028 8.929293   0
```

```
tail(pred_exp, 3)
```

```
     Estimate Est.Error     Q2.5    Q97.5     Base Trt
198  95.87775  26.57965 54.69517 157.3572 148.0707   1
199  99.55114  27.92644 56.40949 164.2998 149.5354   1
200 103.36493  29.33810 58.15550 171.4156 151.0000   1
```

The reason for me being particular about this is that the transformed posterior summary will not, in general, be equal to the summary of the transformed posterior draws. Since we aim for an accurate representation of a quantity's posterior, we always want to see the posterior first, which in our case means getting draws from it. And afterwards, we can summarize as we please. Let's illustrate this point and compute summaries in the "wrong" way:

```
pred_exp_wrong <- pred %>%
  posterior_summary() %>%
  { exp(.) - 1 } %>%
  bind_cols(newdata)

head(pred_exp_wrong, 3)
```

```
  Estimate  Est.Error     Q2.5    Q97.5     Base Trt
1 2.581922 0.09602501 1.996278 3.271859 6.000000   0
2 2.706894 0.09327028 2.109655 3.396354 7.464646   0
3 2.836226 0.09061376 2.234789 3.539028 8.929293   0
```

```
tail(pred_exp_wrong, 3)
```

```
    Estimate Est.Error     Q2.5    Q97.5     Base Trt
198 92.42458 0.3091193 54.69517 157.3572 148.0707   1
199 95.88295 0.3133158 56.40948 164.2998 149.5354   1
200 99.46934 0.3175286 58.15549 171.4156 151.0000   1
```

Let's carefully compare each of the summary statistics. For the posterior mean results are different, sometimes by more sometimes by less, but they overall seem to be roughly in the same ballpark (more on this below). The posterior SD however, is completely different, for our data by a factor of about 3 to 4! Both the mean and SD fall under the category of expectation-based summaries. For them, we have Jensen's inequality to tell us that, for a convex function $g(x)$, e.g., $g(x) = \exp(x) + 1$, we have

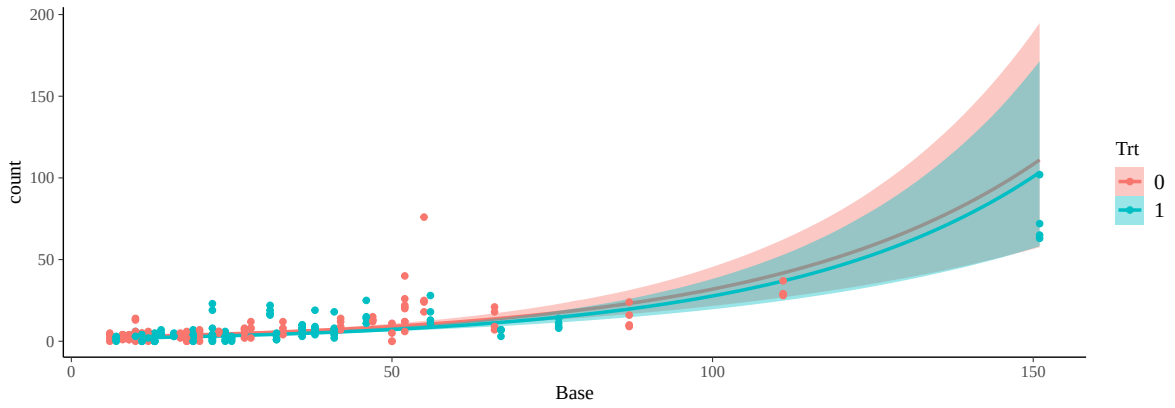
$$g(\mathbb{E}(x)) \leq \mathbb{E}(g(x))$$

This inequality, we see clearly represented in the differing results for posterior mean and SD. The quantiles on the other hand seem to not care about the order of transformation and summary: We say they are *equivariant* under these operations. There is a mathematical rule also behind this result, which says that quantiles are equivariant under any monotonically increasing function $g(x)$, e.g., $g(x) = \exp(x) + 1$:

$$g(Q(x)) = Q(g(x))$$

where I am using $Q(x)$ to denote an arbitrary quantile extract from the variable x. Finally, after this little rant, let's plot the predictions:

```
pred_exp %>%
  ggplot(aes(Base, Estimate, color = Trt, fill = Trt,
             ymin = Q2.5, ymax = Q97.5)) +
    geom_smooth(stat = "identity") +
    geom_point(data = epilepsy, aes(Base, count, color = Trt),
               inherit.aes = FALSE) +
    ylab("count")
```

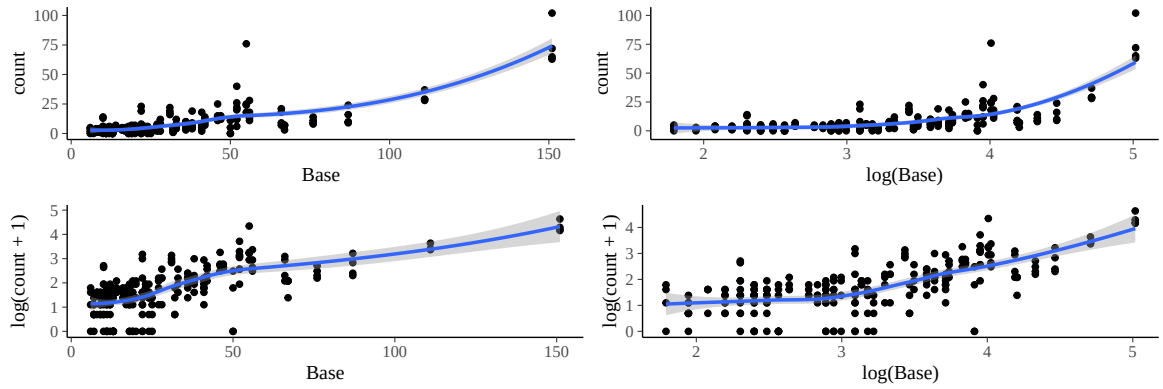


Despite our efforts, these transformed predictions do not represent samples for the mean of the posterior predictive distribution on the original scale. On the log-scale, it predictions represented both the mean, the median and the mode at the same time since the predictive distribution of a Gaussian likelihood is symmetric and unimodal. However, after the log transform our results do only represent the median of the posterior predictive distribution on the original scale. Again, this is because the median, and all quantile-measures for that matter are equivariant under monotonically increasing transformations. In contrast, neither the mean nor the mode are equivariant under (non-linear) transformation, as we have seen from the example of Jensen's inequality.

Accordingly, in this simple model, we have seen (non-)equivariance already coming up twice. First, when we discussed transformation and summary of posterior draws. Second, when we discussed transforming means and medians of the posterior predictive distribution. These are actually two different things: We can have draws from the mean or median of the posterior predictive distribution and can then in turn summarize both of these draws by mean or median. I understand this is confusing. Take a moment to reflect on this to make sure the difference is clear.

3.3.2 Log transform both response and baseline counts

As we have seen, the baseline counts (variable `Base`) are a very strong predictor of future counts (variable `count`). But on which scale do we expect them to be related? Above, we have assumed that `Base` is linearily related to `log(count)`. Does that make sense? Let's create some simple scatter plot to get a better feeling for it:



The curves that were put through the points are `loess` curves the default in `geom_smooth` for small datasets. It its not something I would use for inference, but in order to get a quick glance, I think it's nice actually. For more details on this method, see `?stats::loess`. Admittedly, none of the relationships looks particularly linear to me, but the bottom plots predicting log counts at least don't look like a exponential growth. In the original analysis, of the epilepsy dataset (Thall and Vail 1990), the authors used a log score of `Base`, so let's do the same now:

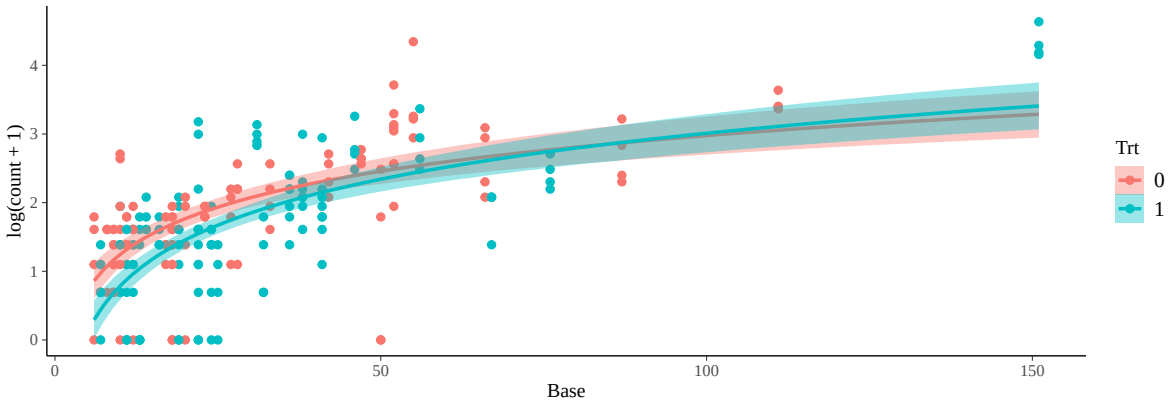
```
fit_epi_gaussian_log2 <- brm(log(count + 1) ~ Trt * scale(log(Base)),
                             data = epilepsy)
```

```
summary(fit_epi_gaussian_log2)
```

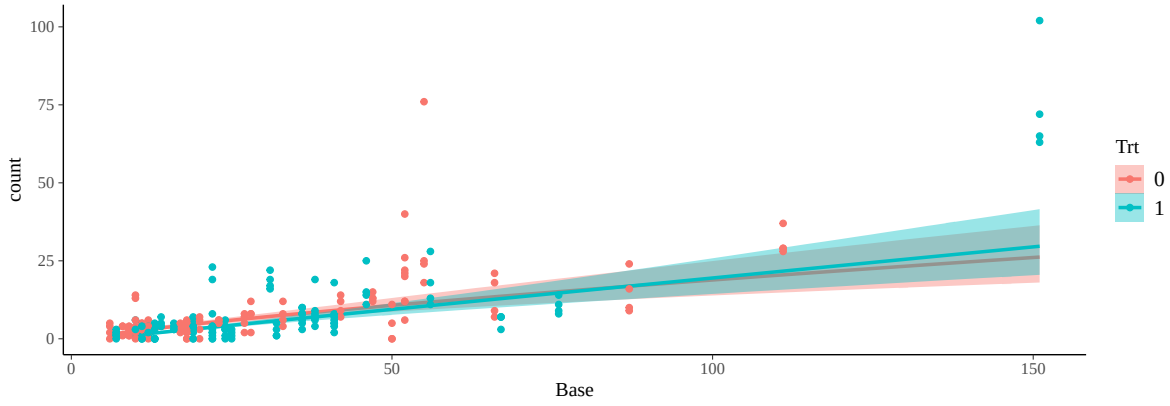
```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: log(count + 1) ~ Trt * scale(log(Base))
   Data: epilepsy (Number of observations: 236)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
                 Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept            1.89      0.07     1.76     2.02 1.00     4340     2858
Trt1                -0.28      0.09    -0.45    -0.10 1.00     4035     2735
scalelogBase         0.56      0.06     0.44     0.68 1.00     3192     3072
Trt1:scalelogBase    0.16      0.09    -0.02     0.34 1.00     3229     2961

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma     0.70      0.03     0.64     0.77 1.00     5233     2619

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

Qualitatively, the overall patterns remain the same. The higher (log) `Base`, the higher the predicted number of seizures, and treatment may have a small effect at least for people with small baseline of seizures. This is confirmed also visually:

```
conditional_effects(fit_epi_gaussian_log2, effects = "Base:Trt")
```



Notice that we still see the untransformed `Base` variable on the x-axis. Accordingly, the relationship is now logarithmic on the log scale of `count`, as a result of assuming a linear relation between log `count` and log `Base`. In order to see the predictions on the original scale of `count`, we again use the manual approach via `posterior_epred` shown above, which gives us the following result:



We see that the relationship of `count` and `Base` on the original scales is now almost linear, but this just a coincidence. The regression coefficients were different, the relationship could indeed be quite non-linear. We can see this by consider the following relationship between to variables x and y that resembles our situation:

$$\log(y) = b_0 + b_1 \log(x)$$

When we now exponentiate both sides and use the math rules of exponential functions, we get:

$$y = \exp(b_0 + b_1 \log(x)) = \exp(b_0) x^{b_1}$$

Accordingly, we relationship will be linear only if the slope parameter `b_1` is not too far from 1. The functional form we have just created is also known as "power law".

Back to our actual modeling task, we can ask which of the two models, using Base or log Base, leads to better predictions:

```
loo_compare(loo(fit_epi_gaussian_log1), loo(fit_epi_gaussian_log2))
```

```
                      elpd_diff se_diff
fit_epi_gaussian_log1  0.0       0.0
fit_epi_gaussian_log2 -2.6       4.6
```

Very similar indeed, certainly not a difference that we can base a decision on. Below, I will continue to use `log(Base)` as predictor for because assuming a power law relationship of baseline and current seizure counts seems like a theoretically sensible choice.

3.3.3 Lognormal models

The combination of Gaussian distribution on a log-transformed variable is so common that it has its own name: the lognormal distribution. To express this family in brms, we can either transform the response variable and then use `family = gaussian()` as we have done above, or use `family = lognormal()` on the "original" (here counts + 1) response:

```
fit_epi_lnorm1 <- brm(count + 1 ~ Trt * scale(log(Base)), data = epilepsy,
                      family = lognormal())
```

When you investigate the results, you will see that they are identical (up to MCMC error) to the corresponding Gaussian model on the log-scale. The advantage of the lognormal approach is that we are automatically getting predictions on the original scale, which makes visualizations and model comparisons easier. For example, if we were to run `conditional_effects`, we would immediately get the plot for the original scale (counts + 1 in our case).

3.3.4 Poisson models

At the start of this chapter, I promised you GLMs, but all I did so far was to play around with different variations of the normal likelihood. Time to make a change. If people ask me what distributions they should consider as likelihoods, I always recommend to use some that are (what I call) *structurally faithful*, by which I mean that it fits naturally to the kind of response being modeled (Bürkner, Scholz, and Radev 2023b). For example, if my responses are lower-bounded counts, I should prefer distributions that are defined exactly for such kind of data. Probability theory and statistics are quite old fields by now, so for almost every imaginable kind of variable, someone has probably thought of corresponding structural faithful

distributions. For lower-bounded counts, the Poisson distribution is the canonical choice (as an initial starting point). For all $y = 0, 1, 2, \ldots, ...$, the Poisson density looks as follows:

$$p(x \mid \lambda) = \frac{\lambda^x}{x!} \exp(-\lambda)$$

It just has one parameter, λ, the mean of the Poisson distribution. In Figure 3.1, you can see a visualization of the Poisson distribution for different λ.

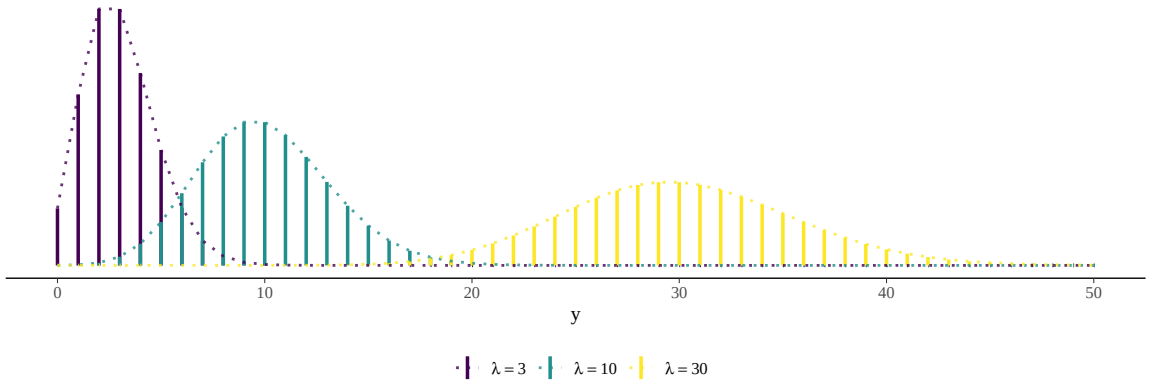


Figure 3.1: Three example densities of the Poisson distribution varying in the mean λ.

Since λ is the mean parameter, it constitutes the ideal prediction target in the context of GLMs. There is only one problem: λ is the mean of counts, so it must necessarily be positive. But a linear predictor $\eta = b_0 + b_1 x_1 + ...$ may take on any real values, also negative ones. So how do we make sure this won't cause use trouble? The answer is that we won't set $\lambda = \eta$ but rather $\lambda = \exp(\eta)$. We can write this equivalently as $\log(\lambda) = \eta$. Accordingly, in comparison to our previous approaches, we do not log transform the response variable directly but rather the mean parameter. We say that we have applied the log *link function*, which links the likelihood's mean parameter to the linear predictor.

The inverse link function, here exp is also called *response function*. I personally find it more intuitive to think in terms of response functions rather than link functions, because the former follows the natural logic of computation. I first compute my linear predictor η, than I apply the response function to obtain λ, then I plug λ into the Poisson likelihood. That being said, in the established Statistics literature, we call these transformation always by their link function name and I guess it's too late to change this habit now. Accordingly, also in brms, we specify the link rather than the response function. Let us run our first Poisson model in brms:

```
fit_epi_poisson1 <- brm(count ~ Trt * scale(log(Base)), data = epilepsy,
                        family = poisson("log"))
```

The main change is the specification of `family = poisson(link = "log")`. We might have also written just `family = poisson()` leading to the same result, because log is the default link for Poisson models. Notice also, that we no longer need the awkward count $+\mathbf{1}$ approach, since we now log transform the Poisson mean $\lambda > 0$, which can never become exactly zero.

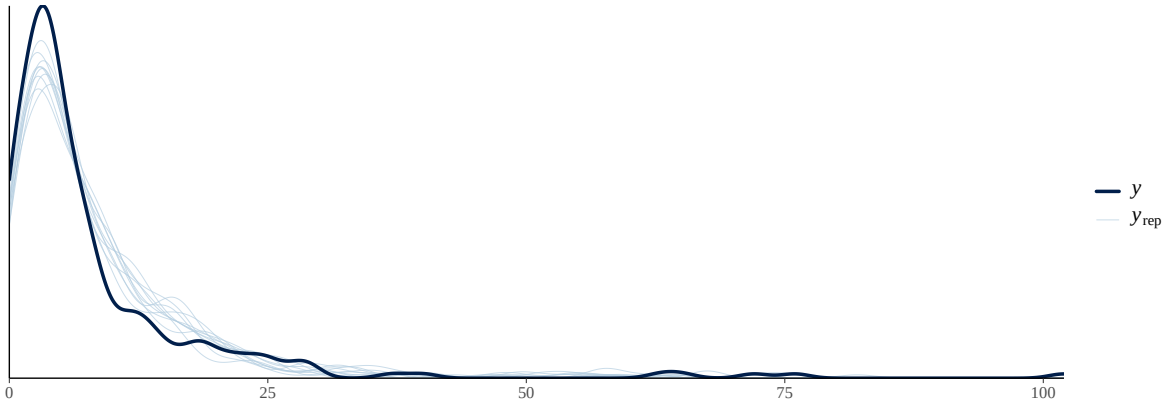
```
summary(fit_epi_poisson1)
```

```
 Family: poisson
  Links: mu = log
Formula: count ~ Trt * scale(log(Base))
   Data: epilepsy (Number of observations: 236)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
                 Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept            1.90      0.04     1.82     1.98 1.00     2068     2745
Trt1                -0.38      0.06    -0.50    -0.26 1.00     1650     2221
scalelogBase         0.72      0.03     0.65     0.78 1.00     1545     2184
Trt1:scalelogBase    0.31      0.05     0.22     0.40 1.00     1531     1895

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

As opposed to the lognormal model from earlier on, the Poisson model suggests a substantial interaction effect of baseline seizures and treatment. Not only is the posterior mean about twice as large, but also the posterior standard deviation is about twice as small. Which of these two results should we rather trust? Let's first approach this question by investigating the Poisson model fit graphically.

```
pp_check(fit_epi_poisson1)
```

```
Using 10 posterior draws for ppc type 'dens_overlay' by default.
```



As you know by now, the default posterior predictive check produces a density, which is of course not an ideal representation for count data, but in the present case, it still does the job. We see that the marginal fit is quite good overall. However, already in this plot, we see seemingly small but, to the experienced eye, clearly visible issues: The bulk of the data distribution close to zero is underestimated by the Poisson model and so is the right tail. There is a specific reason for this pattern: The Poisson distribution has just a single parameter λ, which happens to be not only the mean but also the variance of the distribution; a property called *equidispersion*. However, most real-world count data is actually *overdispersed*, that is, has higher variance than mean, something that the Poisson distribution is not able to handle.

In the context of regression models, we should perhaps better speak of *conditional equidispersion* and *conditional overdispersion*, because what matters are the data distribution after conditioning on the predictors. This is actually a very important point that I think can barely be stressed enough regardless of the statistical framework you fit our model in: The likelihood distribution should be appropriate for the noise in the responses that remains after conditioning on the predictors. Using normal linear regression as an example, we do not need to care if the marginal (i.e. unconditional) response distribution is actually normal, but if the (conditional) distribution of the residuals is normal. The same holds for all kinds of other likelihood distributions, including Poisson models. While the above posterior predictive check shows only the marginal distribution, if the conditional distributions are overdispersed, we will likely see this in the marginal too.

3.3.5 Negative binomial models

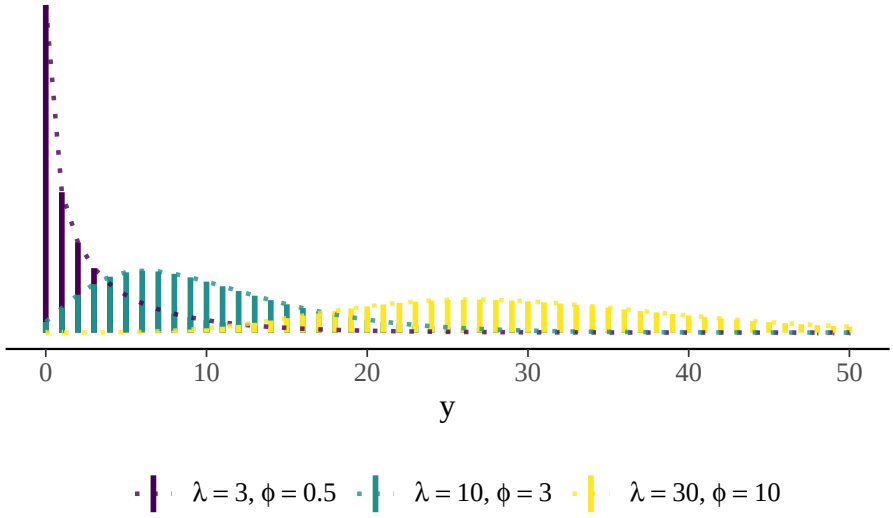
So how can we deal with conditional overdispersion in count data models? The negative binomial distribution provides a popular answer. It has two parameters and a popular paramerization that is commonly used for the purpose of GLMs: One parmaeter is still being the mean $\lambda > 0$ that we also know from Poisson and a second is a shape (precision) parameter $\phi > 0$ which controls the amount of overdispersion. As ϕ increases towards infinity, the negative binomial distribution approaches equidispersion, and thus the Poisson distribution. As such the

negative binomial is a generalization of Poisson. The density of the negative binomial (in the parameterization used for GLMs) is quite involved, and I show it hear just for completeness:

$$p(y \mid \lambda, \phi) = \binom{x + \phi - 1}{x} \left(\frac{\lambda}{\lambda + \phi}\right)^x \left(\frac{\phi}{\lambda + \phi}\right)^\phi$$

To be honest, I find it hard to imagine that this density becomes the Poisson as $\phi \to \infty$ but I guess that is the beauty of math and limiting cases, in which interesting things do happen. Let's get a better intuition by visualizing some example of the negative binomial density:

```
plot_dist("nbinom", c(0, 50),
          pars = list(c(mu = 3, size = 0.5), c(mu = 10, size = 3),
                      c(mu = 30, size = 10)),
          parnames = c("\\lambda", "\\phi"), xtype = "d")
```



In brms, we specify a negative binomial likelihood via `family = negbinomial()`. As for the Poisson, the default link function is `"log"` so if we don't specify a link ourselves, we will get the log link anyway. We are now ready to fit our first negative binomial model in brms:

```
fit_epi_negbin1 <- brm(count ~ Trt * scale(log(Base)), data = epilepsy,
                       family = negbinomial())
```

```
summary(fit_epi_negbin1)
```

```
 Family: negbinomial
  Links: mu = log; shape = identity
Formula: count ~ Trt * scale(log(Base))
   Data: epilepsy (Number of observations: 236)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
                  Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept             1.91      0.07     1.77     2.06 1.00     4445     3065
Trt1                 -0.31      0.11    -0.52    -0.10 1.00     4385     2871
scalelogBase          0.68      0.07     0.56     0.81 1.00     3058     2954
Trt1:scalelogBase     0.20      0.10     0.01     0.39 1.00     3145     2988

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
shape     2.55      0.35     1.94     3.30 1.00     4304     2973

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

Let us check the results. First of all, we see that the model seems to have converged well, for some parameters even with super efficiency, i.e., ESS higher than the total number of draws. The posteriors of the regression coefficients look qualitatively similarly than the ones of the Poisson model. That is, the coefficient of the treatment (for average baseline scores) is negative, the coefficient of baseline (for the control group) is strongly positive, and the interaction coefficient is positive. That said, both the posterior mean and the posterior uncertainty vary clearly between negative binomial and Poisson: The former not only has smaller coefficients on average (posterior means closer to zero), but also larger posterior uncertainty (larger posterior standard deviations and credible intervals). This together implies that the negative binomial model is far less certain about the predictive value of the covariates than the Poisson model. But why is this the case?

Let's investigate the innocent looking `shape` parameter at the bottom of the summary output. It's mean is around 2.57 with a credible interval between roughly 1.94 and 3.33. That doesn't tell us much admittedly. We only know that if `shape` would be infinite, the negative binomial and Poisson distribution would be the same. But how far is 2.57 away from infinity in our case? That is, how can we interpret the `shape` parameter? For this purpose, it is useful to investigate the relationship of the `shape` parameter ϕ with the variance of the negative binomial distribution:

$$\text{Var}(y) = \lambda + \lambda^2/\phi$$

From the above formula for the variance, we have already seen that the impact of ϕ on the variance also depends on the mean λ. So let's take a look at the the ratio of variance and mean for different combinations of λ and ϕ, as shown in Figure 3.2. On the y-axis, we see the variance-mean ratio, that is:

$$\frac{\text{Var}(y)}{E(y)} = \frac{\lambda + \lambda^2/\phi}{\lambda} = 1 + \lambda/\phi.$$

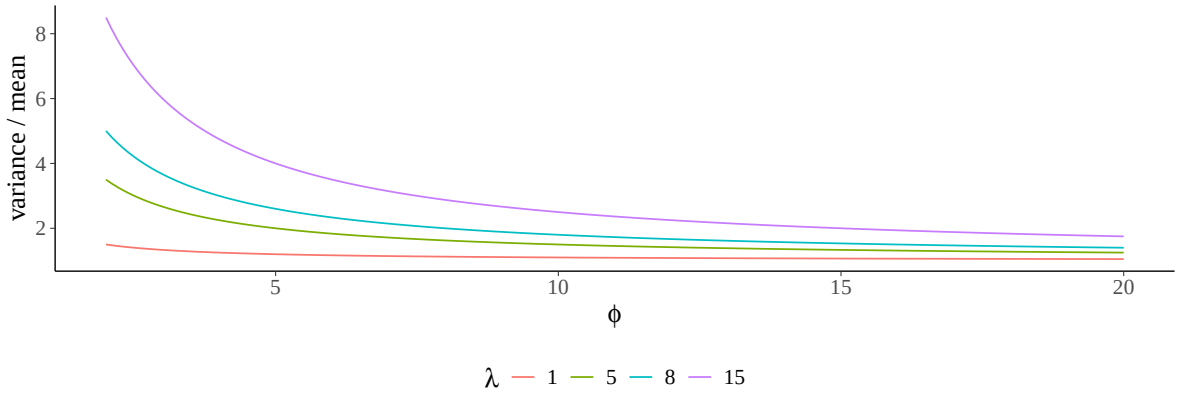


Figure 3.2: Variance-mean ratio of the negative binomial distribution as a function of the mean λ and shape ϕ.

Based on the formula and as also visible in the figure, we see that the variance-mean ratio converges to 1 (i.e., to equidispersion) for $\phi \to \infty$. We also see that for small ϕ values, the ratio can be substantially bigger than zero, especially if the mean λ is not very small. To make this more concrete, let us take a look what ratio we can expect for our model. First, we use a quick and dirty approach by using the exponential transformed posterior mean of `Intercept` $\approx \exp(1.91) \approx 6.75$ as λ (i.e., the mean when all predictors are set to zero) and the posterior mean of `shape` ≈ 2.57 as ϕ. Then we find the ratio to be $1 + \lambda/\phi = 1 + 6.75/2.57 = 3.62$. Accordingly, variance-mean ratios of around 3 to 4 is not unrealistic for our negative binomial model; ratios that are indeed vary far away from what Poisson would expect (i.e., a ratio of 1).

For the purpose of practicing working with posterior draws, let us check out the overdispersion a bit more systematically for our model. First, we will extract the draws of the posterior predictive means for each observation as well as the draws from `shape`

```
draws_pp_means <- posterior_epred(fit_epi_negbin1)
str(draws_pp_means)
```

```
 num [1:4000, 1:236] 3.33 3.46 4.06 3.5 3.7 ...
```

```
draws_shape <- as.data.frame(fit_epi_negbin1, variable = "shape")$shape
str(draws_shape)
```

```
 num [1:4000] 2.98 2.25 2.78 2.91 2.91 ...
```

The `draws_pp_means` matrix has on column per observation and one row per posterior draw. The `draws_shape` vector has one element per posterior draw, since we assumed `shape` to be constant across observations. R stores matrices in column major order. That is, when we compute transformations involving both matrices and vectors, the vector is implicitly extended to match the size of a matrix, in a way that the first column is filled first, then the second column, and so on. In other words, if we work with vectors, matrices, or arrays where the first dimension (i.e., the rows) are indicating the draws, we can just combine matrices, array, and vectors in the same vectorized R expression, provided that their number of draws match. Suppose I want to get the posterior means of the variance-mean ratio for each observation. Then I can write:

```
draws_var_mean_ratios <- 1 + draws_pp_means / draws_shape
means_var_mean_ratios <- colMeans(draws_var_mean_ratios)
str(means_var_mean_ratios)
```

```
 num [1:236] 2.36 2.36 1.78 2.02 8.03 ...
```

A histogram of these posterior means can be found in Figure 3.3. Indeed, there are a lot of observations with a variance-mean ratio higher than 3. One observation even has a ratio of about 20.

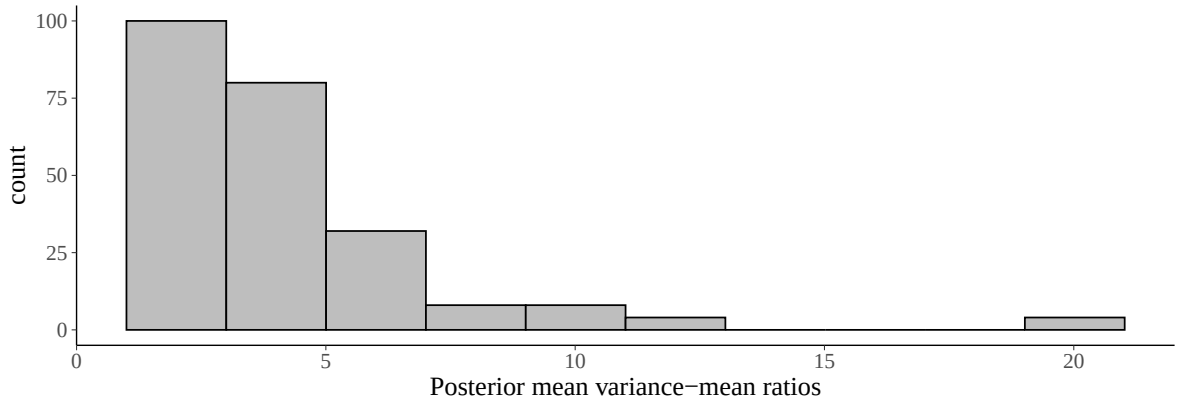


Figure 3.3: Histogram of posterior mean variance-mean ratios per observation of the negative binomial model.

All of the above analyses suggest substantial overdispersion in the data. This in turn suggests that the negeative binomial family is a better choice than the Poisson family. To be fair, just looking at the `shape` parameter in the summary output makes this pretty clear, but I wanted to show you these additional analyses to help you understand my rational better. Let's see if LOO-CV agrees with our conclusion.

```
loo_epi_poisson1 <- loo(fit_epi_poisson1)
loo_epi_negbin1 <- loo(fit_epi_negbin1)
```

```
loo_epi_poisson1
```

```
Computed from 4000 by 236 log-likelihood matrix.

         Estimate    SE
elpd_loo   -864.6  79.8
p_loo        22.1   7.0
looic      1729.2 159.5
------
MCSE of elpd_loo is NA.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.4, 1.1]).

Pareto k diagnostic values:
                         Count Pct.    Min. ESS
(-Inf, 0.7]   (good)     235   99.6%   130
   (0.7, 1]   (bad)        1    0.4%   <NA>
   (1, Inf)   (very bad)   0    0.0%   <NA>
See help('pareto-k-diagnostic') for details.
```

While computing approximate LOO-CV, we encountered the above warning. It suggests that PSIS failed for one observation in the Poisson model (Pareto $\widehat{k} > 0.7$). A closer inspection reveals that one of these problematic observation number 49 containing the largest observed count in the data with `count = 102`. This observation we already saw poorly predicted in the posterior predictive check earlier. This alone points to some potential misfit of the Poisson model. And, since the warning didn't occur for the negative binomial model, it provides some further evidence that the latter is more appropriate. Remember from Chapter X that the Pareto $\widehat{k}$ has a double interpretation in the context of LOO-CV. First, it tells us how well PSIS worked to approximate the LOO-posterior for the full posterior of a given observation. Second, it tells us how influential individual observations are for the posterior. And our observation number 49 seems to be quite influential indeed. Anyway, if we set the issue with this observation aside for a second, what does the LOO-based model comparison currently tell us?

```
loo_compare(loo_epi_poisson1, loo_epi_negbin1)
```

```
                elpd_diff se_diff
fit_epi_negbin1     0.0       0.0
fit_epi_poisson1 -208.1      65.6
```

Indeed, and unsurprisingly in light of what we saw before, the negative binomial model is overwhelmingly better, both in absolute ELPD points and relatively to the corresponding standard error. This comparison is certainly biased because PSIS failed badly for a single observation, but the difference is so brutal that a single failed PSIS is highly unlikely to explain it. Accordingly, we could in good faith just leave it at that and go with the negative binomial model. But out of curiosity, let's fix PSIS by running moment matching:

```
loo_epi_poisson1_mm <- loo_moment_match(fit_epi_poisson1, loo_epi_poisson1)
loo_epi_poisson1_mm
```

```
Computed from 4000 by 236 log-likelihood matrix.

         Estimate    SE
elpd_loo   -864.9  79.8
p_loo        22.4   7.3
looic      1729.9 159.7
------
MCSE of elpd_loo is 0.1.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.4, 1.1]).

All Pareto k estimates are good (k < 0.7).
See help('pareto-k-diagnostic') for details.
```

Indeed, it seemed to have worked as now all Pareto $\hat{k} < 0.7$. Let's compare the two models again:

```
loo_compare(loo_epi_poisson1_mm, loo_epi_negbin1)
```

```
                elpd_diff se_diff
fit_epi_negbin1     0.0       0.0
fit_epi_poisson1 -208.4      65.7
```

There seems to be barely any difference to the comparison before moment matching, but there might as well have been a much bigger one. So please, take high Pareto $\hat{k}$ values seriously, especially if there are several of them and the difference between models is not as huge as it was in our current case. With all this evidence, we are finally able to let the Poisson likelihood go and choose to move forward with negative binomial models.

I have worked with many unbounded count responses in the past, and the negative binomial model was almost always better than the corresponding Poisson model. So why is real data so often overdispersed? If the events come in over time at a certain constant rate, the Poisson distribution arises naturally. That is why some people used to see very clean data, e.g., in some parts of physics, will equate any unbounded count process with the Poisson distribution. But medical, psychology, or biological data from living beings is rarely that clean. Even if measurement itself is relatively precise, living beings especially humans or other higher developed specifies tend to vary quite strongly from individual to individual, and even within the same individual over time. This variation is almost impossible to fully account for by means of the predicting variables we have access to. Any unaccounted within- or between-individual variation will necessarily become an additional source of noise from the perspective of the model. This noise is what causes the overdispersion.

3.3.6 More adventures into model comparison

While reading this chapter, you may have asked yourself already why we didn't compare the lognormal models fitted at the start to the Poisson and negative binomial models. The reason is that the former have a continuous likelihood distribution while the latter have a discrete likelihood distributions. So, for the former, probabilities are integrals under the density, while for the latter, probabilities are the density values themselves. So really quite something different.

But is it valid to compare continuous and discrete likelihood models using density-based metrics such as ELPD? The short answer is *no*. The long answer is that it actually works approximately, at least in certain cases. If you are happy with the short answer and had enough of math today, you can safely skip this section and move right to the next case study in Section 3.4.

For those of you who stayed, let's just check the LOO comparisons for a start:

```
loo_epi_lnorm1 <- loo(fit_epi_lnorm1)
loo_compare(loo_epi_lnorm1, loo_epi_poisson1_mm, loo_epi_negbin1)
```

```
Warning: Not all models have the same y variable. ('yhash' attributes do not
match)
```

```
                 elpd_diff se_diff
fit_epi_negbin1      0.0       0.0
fit_epi_lnorm1      -8.8       5.8
fit_epi_poisson1 -208.4      65.7
```

We get a warning that the response variable is not the same across models. This is not because of the continuous vs. discrete situation but because we have added 1 to the count to make them usable with the lognormal likelihood. If we take these results seriously for a second, we would conclude that the lognormal model is perhaps slightly worse than the negative binomial model but substantially better than Poisson. But can we trust these conclusions? If we remember the discussion about ELPD from Chapter X, we see that it is built on log-likelihood density values. For LOO-CV via ELPD to be fair, the log-likelihood density values need to be comparable. And for the latter to be the case, they need to be on the same scale. For discrete densities, that is easy since we just have to make sure that the density values sum to 1. Let's verify this for the negative binomial model, by exemplary summing over the density values of count 0 to 500 for a person in the treatment group with a Base seizure count of 5:

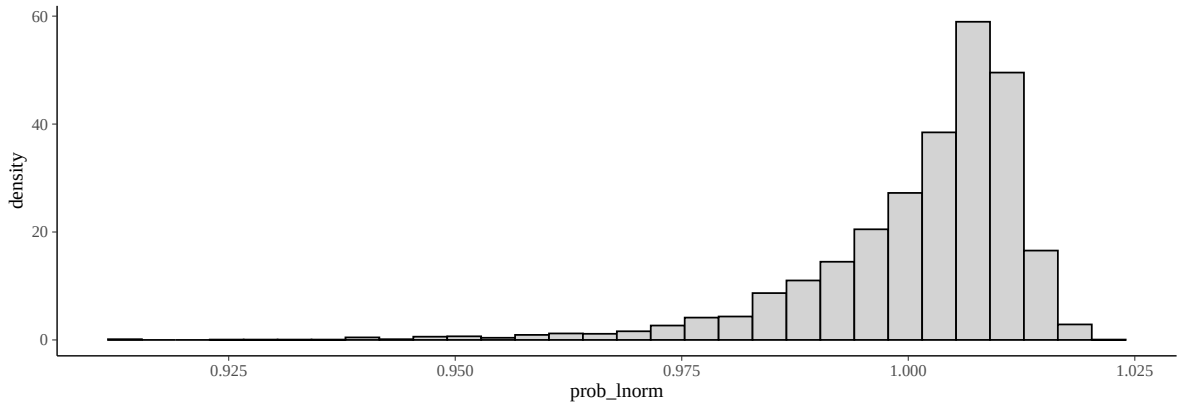
```
dat_all_base <- data.frame(Trt = 0, Base = 5, count = seq(0, 500, 1))
```

```
prob_negbin <- log_lik(fit_epi_negbin1, newdata = dat_all_base) %>%
  exp() %>%
  rowSums()
```

All of these probabilities evaluate to 1, more precisely almost 1 since we didn't add the vanishingly small probability of predicting seizure counts over 500. Now, let's do the same with the lognormal model:

```
prob_lnorm <- log_lik(fit_epi_lnorm1, newdata = dat_all_base) %>%
  exp() %>%
  rowSums()
```

```
data.frame(prob_lnorm = prob_lnorm) %>%
  ggplot(aes(prob_lnorm, after_stat(density))) +
  geom_histogram(fill = "lightgrey", color = "black", bins = 30)
```



Okay, so not exactly 1 but actually quite close for most draws. First of all, why not exactly 1? As mentioned before, for continuous densities, probabilities are defined as integrals under density not as sums of density values at a bunch of discrete points. So there is no reason to expect such a sum to be 1 exactly. And yet we found sums which are actually quite close to 1. Was that just luck or is there actually something systematic behind it?

What we did was to evaluate the lognormal distribution only at the positive natural numbers $k = 1, 2, ...$ and then sum over the resulting density values. Since these values are equidistantly placed with a distance of one, we can turn these discretized density values into a rectangles of width 1 with height equal to the density values $p(k)$. For $k = 1$ the rectangular goes from $y = 0.5$ to $y = 1.5$, for $k = 2$ it goes from $y = 1.5$ to $y = 2.5$, and so on. Now, we can interpret $p(k)$ as approximating the average density in the whole surrounding interval $p(k) \approx \int_{k-0.5}^{k+0.5} p(y)dy$. As a result, we get

$$\begin{aligned}
\sum_{k=1}^{\infty} p(k) &\approx \sum_{k=1}^{\infty} \int_{k-0.5}^{k+0.5} p(y)dy \\
&= \int_{0.5}^{\infty} p(y)dy \\
&= 1 - \int_{0}^{0.5} p(y)dy \\
&\approx 1.
\end{aligned}$$

The second approximation we have made is to ignore all the density between in the interval $[0, 0, 5]$, which is justified as long as the log-mean parameter μ of the lognormal distribution is not very small (see Figure 3.4). This shows that, with a little bit of squinting, using likelihood-density based measures such as ELPD to compare actual count-data models with their continuous approximations can be a valid approach, if we interpret only large differences. With regard to the current case study, I wouldn't use ELPD to choose among the negative

binomial and the log-normal model because their performance is so similar that the small errors induced by the continuous vs. discrete comparison may actually matter. But we can safely conclude that both are substantially better than Poisson.

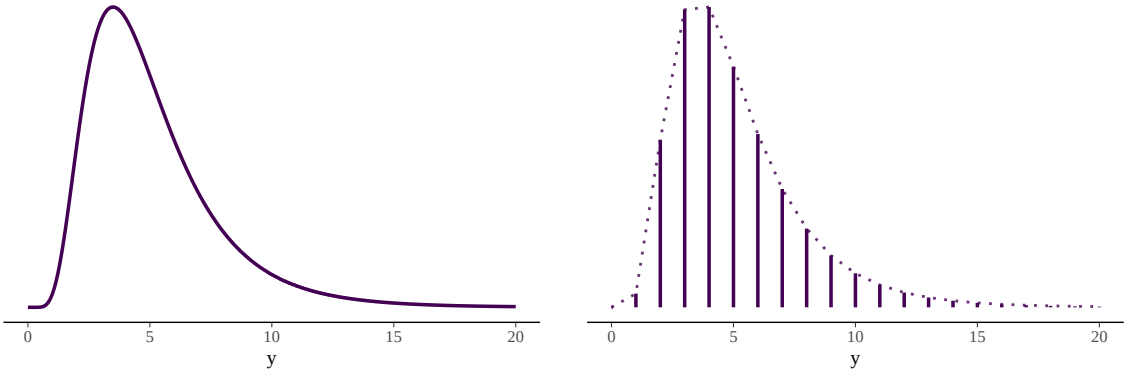


Figure 3.4: Lognormal density with log-mean $\mu = 1.5$ and log-SD $\sigma = 0.5$. Left: Regular density function. Right: Discretized version of the density on the positive natural numbers $(1, 2, ...)$.

With this, let us leave the epilepsy dataset for now. We will come back to it later in the context of multilevel models in Section X.

3.4 GLMs for double-bounded responses

In this section, we will take an initial look at Bayesian GLMs for double-bounded responses, that is, responses that have both an upper and a lower bound. In particular, binary (0-1) data falls within this class and we will focus on them for now. As a case study, we will predict the fate of passengers on the fatal maiden voyage of the famous ocean liner "Titanic". Among others due to insufficient number of live boats, 1502 out of 2224 passengers and crew died in the Titanic accident. Our goal is to figure out which groups of passengers had a higher chance of surviving. For this, we use the dataset shipped with the `titanic` R package (Hendricks 2015), which resembles the one provide in the titanic Kaggle challenge (https://www.kaggle.com/competitions/titanic). For each of the passengers, we have quite a bit of data available, in particular their socio-economic class (variable `Pclass`; coded as first, second, and third class), their `Sex` (a binary indicator; in 1912 we weren't that far in terms of acknowledging sex and gender diversity), their `Age` (in years), as well as some other information, such as the kind of tickets, fare, and the cabin they were staying in (if known). The response variable `Survivad` is 1 if a person survived and 0 if they died.

Survived	Pclass	Sex	Age	Ticket	Fare	Cabin
0	3	male	22	A/5 21171	7.25	
1	1	female	38	PC 17599	71.28	C85
1	3	female	26	STON/O2. 3101282	7.92	
1	1	female	35	113803	53.10	C123
0	3	male	35	373450	8.05	
0	3	male	NA	330877	8.46	

We will primarily focus on the predictors class, sex, and age predictors, the former two we will turn into factors to ensure proper behavior within R formula syntax:

```
titanic <- titanic::titanic_train %>%
  mutate(Pclass = factor(Pclass),
         Sex = factor(Sex))
```

As likelihood, we will choose the Bernoulli distribution. Due to responses being either 0 or 1, it has a remarkably simple form. We say that $y = 1$ (i.e., a person survived the titanic accident) happened with a to-be-estimated probability θ and, accordingly, $y = 0$ (the person died) happens with probability $1 - \theta$. We can write this conveniently into a single equation which constitutes the density of the Bernoulli distribution:

$$p_{\text{Bernoulli}}(y \mid \theta) = \theta^y \, (1 - \theta)^{1-y}$$

Now, we of course don't want to just estimate a single overall θ for the whole dataset. That would be a bit boring. Instead, we want to understand how θ varies between people. For this purpose, we will again set up a linear predictor η that we will connect to θ via an appropriate response function. Since θ is a probability parameter by definition, in can only take on values in the unit interval $[0, 1]$. Accordingly, we need a response function that maps real values to values in the unit interval. Here, we choose the logistic response function, which is the most common choice in most fields:

$$\theta = \text{logistic}(\eta) = \frac{\exp(\eta)}{1 + \exp(\eta)} = \frac{1}{1 + \exp(-\eta)}$$

The corresponding link (inverse response) function is called the *logit* link and is given by:

$$\eta = \text{logit}(\theta) = \log\left(\frac{\theta}{1 - \theta}\right) = \log(\theta) - \log(1 - \theta)$$

First, we want to predict θ by the class and sex of the passengers. We use dummy coding for these predictors with 3rd class and females as reference categories, respectively. This leads us to the following linear predictor for the nth observation:

$$\eta_n = b_0 + b_1 \text{Class1}_n + b_2 \text{Class2}_n + b_3 \text{SexMale}_n$$

$$b_i \sim \text{normal}(0, 3) \quad i = 0, \ldots, 3$$

In brms syntax, this model can be specified as follows:

```
fit_titanic1 <- brm(
  Survived ~ Pclass + Sex,
  data = titanic, family = bernoulli("logit"),
  prior = prior(normal(0, 3), class = "Intercept") +
    prior(normal(0, 3), class = "b")
)
```

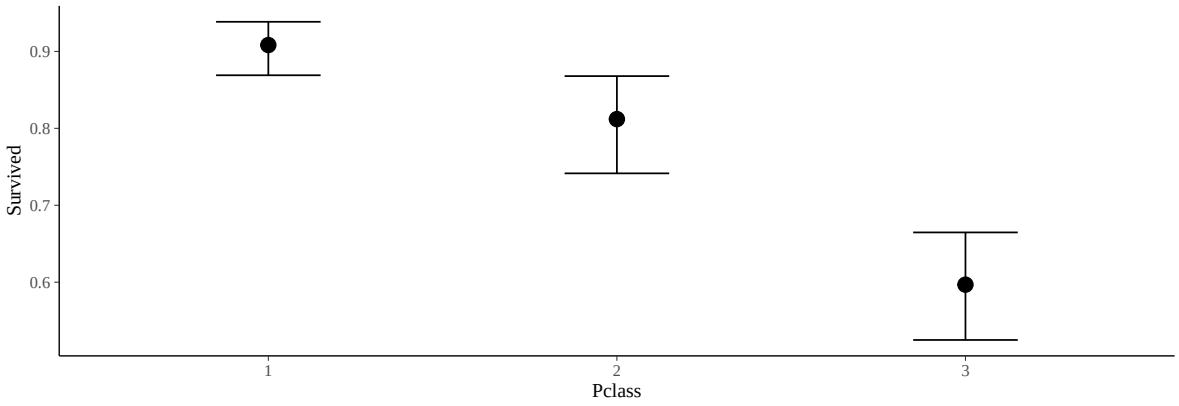
```
summary(fit_titanic1)
```

```
 Family: bernoulli
  Links: mu = logit
Formula: Survived ~ Pclass + Sex
   Data: titanic (Number of observations: 891)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept     2.30      0.21     1.89     2.73 1.00     3412     2685
Pclass2      -0.83      0.24    -1.30    -0.37 1.00     3790     2777
Pclass3      -1.90      0.21    -2.33    -1.50 1.00     3667     2778
Sexmale      -2.65      0.18    -3.02    -2.29 1.00     3623     2915

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

What do we learn from this summary? Looking at the `Pclass` predictor first we see that the regression coefficients of the second and third class – relative to the first class as reference – are clearly negative. This means the probability of survival is clearly lower in the second and third classes than in the first class. But how much lower? This is hard to tell from these numbers alone, since they are all on logit scale. With time you will get a feeling for regression coefficients on logit scales and can tell how big of an effect it roughly corresponds to on the

probability scale. We will learn more about this later on. For now, let it be sufficient to say that an absolute group difference of roughly 2 is quite large already, as we also see visualized from the plot below:

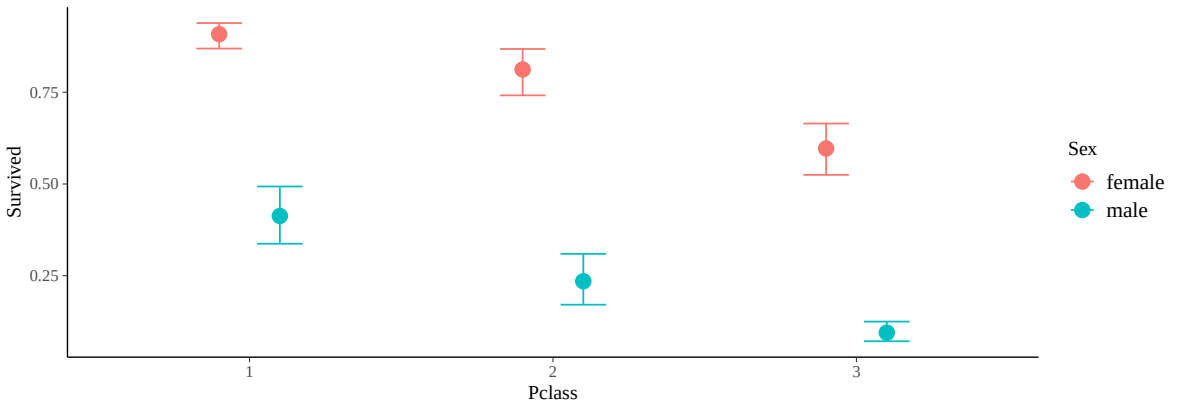
```
conditional_effects(fit_titanic1, "Pclass")
```



The same is true for the difference in survival probabilities between men and women, with man being estimated to have much lower survival probability than women overall. In the summary output, we see this from the clearly negative regression coefficient of `Sexmale`.

We modeled only the main effects of class and sex, so on the logit scale their predictions of the survival probability are only additive by definition. However, since the exponential transform (part of the logistic response function) turns additive relationships into multiplicative ones, as we have already discussed in the context of the count data models in Section 3.3. That is, if we look at the predictions on the response scale, say via `conditional_effects`, we can see the non-linearity of the response function to have an influence.

```
conditional_effects(fit_titanic1, "Pclass:Sex")
```



If the effects of the predictors were purely additive on the response scale, then the lines we could put between through the two blue and red points, respectively, would be exactly parallel to each other. But as you can see, they are not quite parallel. Instead they are "bend" towards the boundaries of $\theta = 0$ or $\theta = 1$, as a consequence of the response function ensuring in a smooth way that predictions will never cross these boundaries. In other words, even if things are linear on the latent scale, a certain amount of non-linearity is unavoidable in GLMs with non-identity link functions.

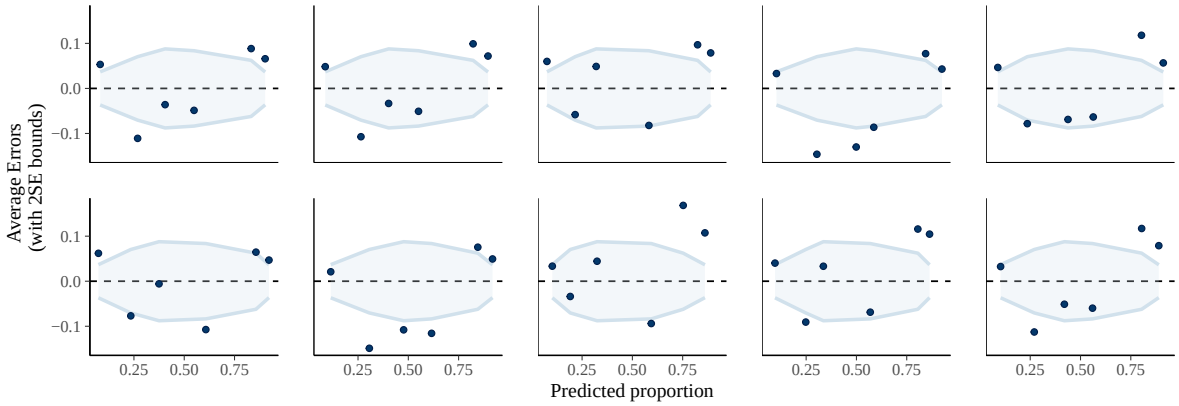
How good does our model fit the data though? Let's start with some posterior predictive checks:

```
pp_check(fit_titanic1, ndraws = 10)
```

Admittedly, this predictive check is a bit pointless. It merely tells us that the model got the ratio of 0s and 1s in the dataset right, something that a Bernoulli model with only an intercept can already do perfectly. More generally, a lot of the standard predictive checks are not sensible for binary responses as the space of model predictions (the unit interval) is different from the space of possible responses (only 0 or 1).

One predictive check that kind of works is the `"error_binned"` type, because it bins responses into groups and then averages them per group. The resulting binned responses, and the corresponding residuals, are now on the same scale as the predictions (i.e., within the unit interval), so comparison becomes sensible:

```
pp_check(fit_titanic1, type = "error_binned", ndraws = 10)
```



Every facet in the above plot corresponds to one posterior draw. Here, we decided to plot 10 draws such that the plot does not get too large. Bins are done according to the predicted proportions. That is, data points that the model assigned similar predictions to will be binned together. We can then check two things: (a) How big are the residuals of the binned responses on the probability scale and (b) how well does the uncertainty in the posterior predictive

distribution reflect these residuals? With regard to (a), we see that the absolute residuals often exceed 10%, a quite substantial prediction error I would say. With regard to (b), the blue shadowed areas should contain roughly 95% of the corresponding points if the model's predictions where well calibrated. But here it seems to be the case for only roughly 50% of the points. In other words, the model is *overconfident*, that is, it underestimates the error in its own predictions. Together, this indicates that we should think about how to further improve our model.

With `fit_titanic1` we have seen that women, overall, survived with a substantially higher probability than men, suggesting that the policy of evacuating women first had worked, at least to some degree. However, we have also seen that second and third class passengers, overall, survived with a substantially lower probability than first class passengers. This begs the question whether the difference in survival probability between woman and men was the same across classes, in other words, whether the interaction between sex and class is predictive of the survival probability.

Before we move on to more complicated models, let's quickly run LOO-CV and store the results so we can reuse it efficiently later.

```
fit_titanic1 <- add_criterion(fit_titanic1, "loo")
loo(fit_titanic1)
```

```
Computed from 4000 by 891 log-likelihood matrix.

         Estimate   SE
elpd_loo   -417.3 17.1
p_loo         3.7  0.2
looic       834.6 34.1
------
MCSE of elpd_loo is 0.0.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.9, 1.2]).

All Pareto k estimates are good (k < 0.7).
See help('pareto-k-diagnostic') for details.
```

The loo summary output shows that PSIS has worked out well (all Pareto k < 0.7). What is more, we see an effective number of parameters `p_loo` $= 3.71$. This coincides nicely with the number of parameters themselves (4), which come in the form of 4 regression coefficients. This also illustrates that our priors where indeed weakly informative. Had we chosen (much) more informative priors, we would have seen `p_loo` to reduce noticeably. Try it out yourself, for example, by assigning highly informative `normal(0, 0.1)` priors to the regression coefficients and then running `loo` again.

3.4.1 Adding interactions

Was the policy of evacuating women first applied with equal success across the different passenger classes? This call for a model including the interaction of class and sex. Leaving everything else the same, our new model looks as follows:

```
fit_titanic2 <- brm(
  Survived ~ Pclass * Sex,
  data = titanic, family = bernoulli("logit"),
  prior = prior(normal(0, 3), class = "b")
)
```

Sampling and convergence are good so we directly check out the posterior of the regression coefficients:

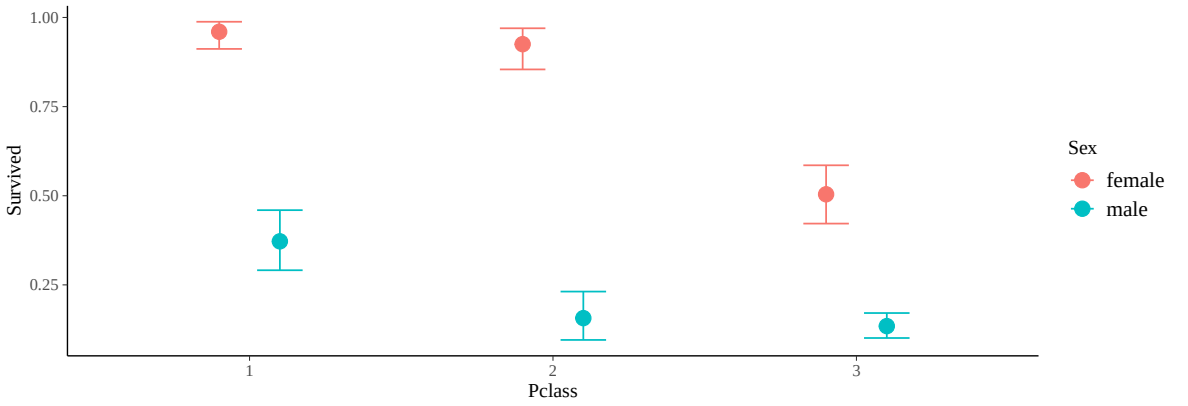
```
summary_table(fit_titanic2)
```

variable	mean	sd	q5	q95
b_Intercept	3.24	0.53	2.46	4.18
b_Pclass2	-0.70	0.68	-1.84	0.39
b_Pclass3	-3.22	0.55	-4.19	-2.40
b_Sexmale	-3.76	0.56	-4.74	-2.91
b_Pclass2:Sexmale	-0.48	0.75	-1.70	0.76
b_Pclass3:Sexmale	1.88	0.60	0.97	2.92

Remember that first class female passengers are the reference category here. In the first class, women had a clearly higher survival rate then men as indicated by the posterior of the main coefficient of `Sexmale` lying strongly below zero. When looking at the posterior of the interaction coefficient related to third class male passengers (coefficient `Pclass3:Sexmale`), we see that it is strongly positive. This means that, compare to first class, the difference in survival rates between women and men was much smaller in the third class, although still in favor of women.

It is much easier to get an understanding of the regression coefficients' meaning once we visualize the predictions:

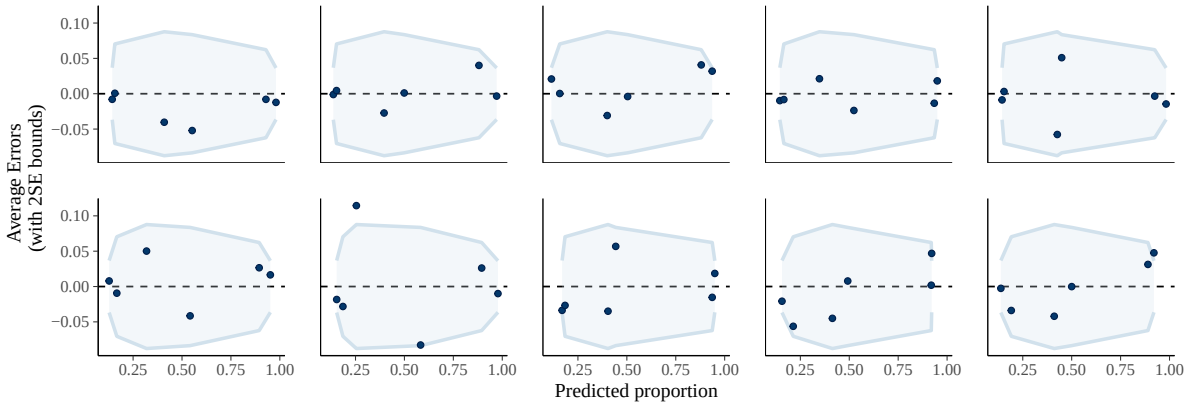
```
conditional_effects(fit_titanic2, "Pclass:Sex")
```



Here we see clearly, and without having to add regression coefficients by hand, that the difference between women's and men's survival is much smaller for the third class passengers than for first and second class passengers. Apparently, the policy of "women first" didn't work out so well in the third class.

Does adding the interaction term improve model fit? Let's first look at some posterior predictive checks:

```
pp_check(fit_titanic2, type = "error_binned", ndraws = 10)
```



This clearly looks much better, than for `fit_titanic1`: The residuals are smaller (mostly below 5%) and the posterior uncertainty seems to be much better calibrated. But in these plots, we still look at in-sample fit, which is prone to overfitting as we know. For a more formal comparison, let's compare the LOO-CV performance:

```
fit_titanic2 <- add_criterion(fit_titanic2, "loo")
loo(fit_titanic2)
```

```
Computed from 4000 by 891 log-likelihood matrix.

         Estimate   SE
elpd_loo   -405.1 17.1
p_loo         5.9  0.6
looic       810.3 34.1
------
MCSE of elpd_loo is 0.1.
MCSE and ESS estimates assume MCMC draws (r_eff in [0.2, 1.1]).

All Pareto k estimates are good (k < 0.7).
See help('pareto-k-diagnostic') for details.
```

```
loo_compare(fit_titanic1, fit_titanic2)
```

```
             elpd_diff se_diff
fit_titanic2   0.0       0.0
fit_titanic1 -12.2       5.2
```

The `loo_compare` results clearly point to an improved out-of-sample predictions of `fit_titanic2` suggesting that the interaction between class and sex is indeed an important aspect of the data to be modeled.

3.4.2 Centering Predictors

Interaction terms are notoriously hard to interpret, as we have to consider them in the context of the main effects terms of the involved variables. In model `fit_titanic2`, we had to consider which reference category was used for both `Pclass` and `Sex` as the interpretation of the interaction coefficients rested on the comparisons with these reference categories. This get awkward pretty quickly as more predictors and their interactions enter the model as we will see in a bit. Let's already come up with a solution now. We know that, for categorical predictors, we will always have to compare against *some* baseline, the question is just which value we choose as that baseline. In the default coding in R, this will be the first category of each predictor, something that we call treatment coding, also known as dummy coding. But what if we would use the mean of all categories instead? The coding that leads to coefficients interpretable this way is called sum coding, also known as effect coding. At least for me, sum coding creates a more intuitive baseline compared to treatment coding, especially once we enter the realm of interaction effects.

Contrast coefficients as a function of group means:

$$\mu = \frac{1}{3}(\mu_1 + \mu_2 + \mu_3) = \frac{1}{3}\mu_1 + \frac{1}{3}\mu_2 + \frac{1}{3}\mu_3$$
$$\delta_2 = \mu_2 - \mu = -\frac{1}{3}\mu_1 + \frac{2}{3}\mu_2 - \frac{1}{3}\mu_3$$
$$\delta_3 = \mu_3 - \mu = -\frac{1}{3}\mu_1 - \frac{1}{3}\mu_2 + \frac{2}{3}\mu_3$$

Group means as a function of contrast coefficients:

$$\mu_1 = 1\mu - 1\delta_2 - 1\delta_3$$
$$\mu_2 = 1\mu + 1\delta_2$$
$$\mu_3 = 1\mu + 1\delta_3$$

If the contrast coefficients are set up as in Equation X, I can compute the group means from the constrast coefficients as per Equation Y. This is a one-to-one relation, so I do this process in the other direction too. That is, if I *want* my contrast coefficients to be those in Equation X, I have to set up the contrast variables as per Equation Y.

```
contrast_matrix <- rbind(
  c(1/3, 1/3, 1/3),
  c(-1/3, 2/3, -1/3),
  c(-1/3, -1/3, 2/3)
)
rownames(contrast_matrix) <- c("mu", "delta2", "delta3")
colnames(contrast_matrix) <- c("mu1", "mu2", "mu3")
```

	mu1	mu2	mu3
mu	1/3	1/3	1/3
delta2	-1/3	2/3	-1/3
delta3	-1/3	-1/3	2/3

```
coding_matrix <- solve(contrast_matrix)
```

	mu	delta2	delta3
mu1	1	-1	-1
mu2	1	1	0
mu3	1	0	1

The above exemplified approach works regardless of the specifically desired contrasts: (1) Set up the contrast equations, that is, the desired contrasts defined as linear functions of the group means. (2) Invert the implied system of linear functions via inversion of the coefficient matrix. (3) Use the inverted matrix to obtain the dummy variables from the original grouping variable. For a most of the common contrast codings, including treatment and sum coding, R provides built-in implementations already, so for them we do not have to make Step (1) and (2) ourselves. What is more, step (3) will be done automatically by R's system to create design matrices from formulas (via the `model.matrix` function), a system that brms also builds upon. That is, all we have to do ourselves it to tell R which contrast coding to apply. The built-in approach works via the `contr.*` assignment functions. For example, we could apply sum coding to Pclass via

```
contrasts(titanic$Pclass) <- contr.sum(3)
```

but this is ugly for the following reasons. First, this syntax isn't nicely integratable into the tidyverse approach to data wrangling to to the `constrasts` function appearing on the left-hand side of the assignment. Second, the unshown category will be the last one for sum coding, but the first one for treatment coding (TODO expand), creating confusing inconsistencies when switching the codings. Third, the dummy variables do not have names, so the resulting dummy variable names will just be X (TODO: check). For these reasons, I always use my own implementation of sum coding, which nicely integrates with the tidyverse, uses the first category as "reference", and creates sensible dummy variables names:

```
# sum (effect) coding in a less awkward way
# codes the first category with -1
# x: categorical vector to be sum coded
# lvls: optional factor levels of x
sum_coding <- function(x, lvls = levels(as.factor(x))) {
  nlvls <- length(lvls)
  stopifnot(nlvls > 1)
  cont <- diag(nlvls)[, -nlvls, drop = FALSE]
  cont[nlvls, ] <- -1
  cont <- cont[c(nlvls, 1:(nlvls - 1)), , drop = FALSE]
  colnames(cont) <- lvls[-1]
  x <- factor(x, levels = lvls)
  contrasts(x) <- cont
  x
}
```

Since the built-in treatment contrast matrix is set up in a nicer way already, my corresponding function for tidyverse integration looks much simpler than that for sum coding:

```
# treatment (dummy) coding in a less awkward way
# codes the first category with 0
# x: vector to be sum coded
# lvls: optional factor levels of x
treatment_coding <- function(x, lvls = levels(as.factor(x))) {
  x <- factor(x, levels = lvls)
  contrasts(x) <- contr.treatment(levels(x))
  x
}
```

With these functions in hand, it is easy to change apply the desired coding schemes to our categorical predictors within a tidyeverse workflow:

```
titanic <- titanic %>%
  mutate(
    Pclass = sum_coding(Pclass),
    Sex = sum_coding(Sex)
  )
```

Let's fit the interaction model again:

```
fit_titanic3 <- brm(
  Survived ~ Pclass * Sex,
  data = titanic, family = bernoulli("logit"),
  prior = prior(normal(0, 3), class = "b")
)
```

As you can see, the `brm` code for `fit_titanic3` is exactly the same as for `fit_titanic2`. The models only differ in terms of the predictor coding, which is implicitly contained only in the `titanic` data frame object.

```
summary(fit_titanic3)
```

```
 Family: bernoulli
  Links: mu = logit
Formula: Survived ~ Pclass * Sex
   Data: titanic (Number of observations: 891)
  Draws: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup draws = 4000
```

```
Regression Coefficients:
                Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept           0.33      0.15     0.07     0.65 1.00     1813     2266
Pclass2             0.08      0.21    -0.31     0.50 1.00     2348     2362
Pclass3            -1.26      0.16    -1.60    -0.97 1.00     1955     2483
Sexmale            -1.70      0.14    -2.00    -1.44 1.00     1802     2095
Pclass2:Sexmale    -0.42      0.20    -0.82    -0.01 1.00     2531     2605
Pclass3:Sexmale     0.77      0.16     0.48     1.11 1.00     1706     2122

Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
and Tail_ESS are effective sample size measures, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

What are the coefficients' posteriors telling us? In terms of the main effects of class, we see that 3rd class has a substantially smaller survival probability than the grant mean across the classes, while the 2nd class is pretty much right on the grant mean. We don't see the coefficient of the 1st class directly in the output but we will compute in a bit. Of course, interpretation of the coefficients' size remains difficult because we are working on the logit scale. Investigation of the main effect of sex, tells use that men have substantially smaller survival probability than the grant mean across men and women. Notice that the coefficient is about half of the one we found in `fit_titanic1`, that is, the model with treatment coding and no interactions. This is no coincidence. Since the grant mean is here just the mean of two groups, the doubled difference between each of the groups and the grant mean is equal to the difference between the two groups. Lastly, in terms of the interactions, we see a negative interaction effect of 2nd class males and a positive interaction effects of 3rd class males. If we view this in relation to the strongly negative main effect of males, this means that the difference in survival probability between males and females becomes even stronger than average in the 2nd class and less strong (but still substantial) in the 3rd class.

Sum coding has the advantages that the intercept always represents the grant mean over groups (more precisely: the mean over group means on the link scale), and thus the predictor-specific coefficients can be interpreted as deviations of the groups from the grant mean. However, we still had to "sacrifice" one level whose coefficient does not immediately show up in the summary. In `fit_titanic3`, for example, we got the coefficients for `Pclass = 2` and `Pclass = 3` (which we called δ_2 and δ_3) but not for `Pclass = 1` (which we will now call δ_1). But we can still compute δ_1 from the model after the fact. By construction of sum coding, the δ coefficients need to sum to zero: $\delta_1 + \delta_2 + \delta_3 = 0$, otherwise the intercept wouldn't be the grant mean. Accordingly, we have $\delta_1 = -\delta_2 - \delta_3$ or, formulated as a per-draw operation $\delta_1^{(s)} = -\delta_2^{(s)} - \delta_3^{(s)}$. That is, we could extract the posterior draws of δ_2 and δ_3 from the model and do the transformation ourselves. Or, if we are lazy, we just use a brms method particularly built for much tasks:

```
hypothesis(fit_titanic3, "- Pclass2 - Pclass3 = 0")
```

```
Hypothesis Tests for class b:
              Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio
1 (-Pclass2-Pclass3) = 0     1.18      0.24     0.76     1.67         NA
  Post.Prob Star
1        NA    *
---
'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
'*': For one-sided hypotheses, the posterior probability exceeds 95%;
for two-sided hypotheses, the value tested against lies outside the 95%-CI.
Posterior probabilities of point hypotheses assume equal prior probabilities.
```

In the first and only row of the returned table, we see the summary of the posterior of δ_1. Clearly, $\delta_1 > 0$ almost surely showing us again that, in comparison to passengers in lower classes, passengers in the first class had a much higher chance of survival.

This also mean that if we change our mind and suddenly prefer our treatment coded coefficients, we could compute it from the sum coded coefficients. For example, if the first class was the desired reference category, and we wanted the contrast it with the second and third classes, we would want to compute

$$\begin{aligned}\mu_2 - \mu_1 &= (\mu + \delta_2) - (\mu + \delta_1) = \delta_2 - \delta_1 \\ &= \delta_2 - (-\delta_2 - \delta_3) = 2\delta_2 + \delta_3\end{aligned}$$

and analogously with $\mu_3 - \mu_1$. Thus to obtain the treatment coded coefficients via `hypothesis`, we can write:

```
hyps <- c(
  "2 * Pclass2 + Pclass3 = 0",
  "2 * Pclass3 + Pclass2 = 0"
)
hypothesis(fit_titanic3, hyps)
```

```
Hypothesis Tests for class b:
               Hypothesis Estimate Est.Error CI.Lower CI.Upper Evid.Ratio
1 (2*Pclass2+Pclass3) = 0    -1.10      0.41    -1.91    -0.29         NA
2 (2*Pclass3+Pclass2) = 0    -2.45      0.35    -3.20    -1.84         NA
  Post.Prob Star
1        NA    *
2        NA    *
```

```
---
'CI': 90%-CI for one-sided and 95%-CI for two-sided hypotheses.
'*': For one-sided hypotheses, the posterior probability exceeds 95%;
for two-sided hypotheses, the value tested against lies outside the 95%-CI.
Posterior probabilities of point hypotheses assume equal prior probabilities.
```

These are still not the coefficients that we got in `fit_titanic2` because there, both predictors (class and sex) were treatment coded while here, we still (implicitly) used the sum coding for sex. We could go ahead and transform the complete set of sum coded coefficients to the complete set of treatment coded coefficients (or vice versa). It's just a bit more math and I leave this for you as an exercise, if you like. What I am trying to say here is that the different coding are *equivalent* in the sense that we can transform their set of coefficients in a one-to-one manner. This means we can in principle freely choose our coding used during model fitting and obtain any other coding afterwards by transforming the posterior draws.

Practically, two things stand in the way though. The first is the math itself and it's easy to get the transformation wrong somewhere. So it is usually just safer to use a coding that already contains the contrasts of interest as parameters or at least allows them to be computed easily during the post-processing. Second, and more Bayesian specific, priors break the equivalence of the different codings. That is, if we choose informative priors in one coding (i.e., in one parameterization), then the resulting transformed coefficients of another cording are in general no longer the same as the coefficients we would have obtained by directly going for that other coding. Accordingly, another way to choose our preferred coding is to think about what parameter we can easily specify priors on. Do we understand specific group difference well, or is it easier to a prior judge the deviation of groups from the overall mean? Usually, in my experience, the parameters that I am interested in are also those I will most likely have prior knowledge on (if I have any). As such, both the math and the prior argument would point us in the same direction.

In our example here, we did specify wide priors, for the logit scale at least, so for practical purposes, we can consider the results arising from our different coding as equivalent. In fact, if we ran some model comparison via, say, LOO-CV we would see that the two models with treatment and sum coding respectively would provide the same predictions up to MCMC error.

3.4.3 Even More Interactions

Another variable which we expect to affect survival rates is the passengers' age, as children were supposed to be prioritized in the evacuation. So let's include age (in years) and model it directly in interaction with class and sex. For this purpose, it makes sense that transform the `Age` variable to simplify interpretation. First, we will divide age by 100 such that it is roughly on a 0 to 1 scale. This ensures that age-related coefficients on the logit scale do not become too small. As an exercise, fit the model below with the unscaled Age variable and check how

small the coefficients will become. Second, we will center age around its mean, such that a centered Age value of 0 will indicate the age mean. In a way, mean centering is "the sum coding for continuous predictors", as it allows us to interpret coefficients as deviations from the predictor's mean.

```
titanic <- titanic %>%
  mutate(
    Age100 = Age / 100,
    Age100c = Age100 - mean(Age100, na.rm = TRUE)
  )
```

The model is then specified as follows:

```
fit_titanic4 <- brm(
  Survived ~ Pclass * Sex * Age100c,
  data = titanic, family = bernoulli("logit"),
  prior = prior(normal(0, 3), class = "b")
)
```

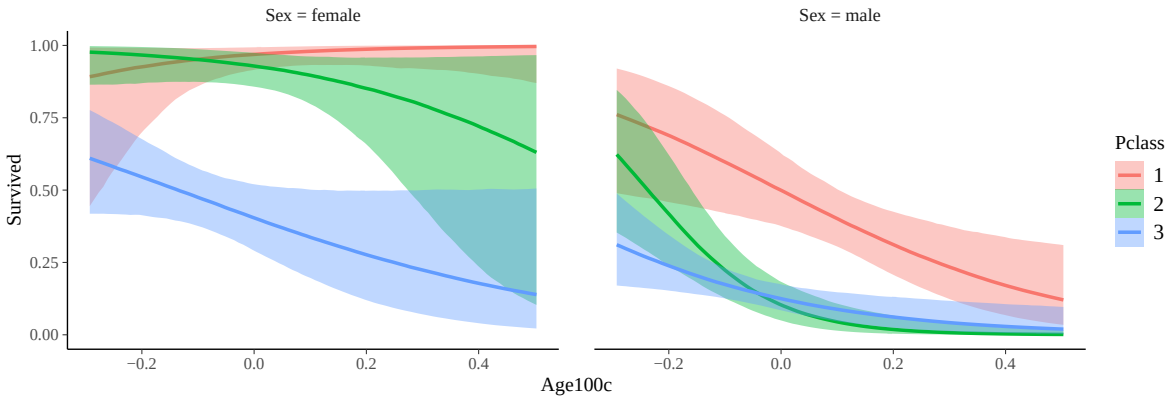
The summary table of coefficients is now quite crowded, with any up to 3-way interaction coefficients:

variable	mean	sd	q5	q95
b_Intercept	0.26	0.17	0.01	0.55
b_Pclass2	-0.05	0.24	-0.45	0.34
b_Pclass3	-1.44	0.19	-1.76	-1.13
b_Sexmale	-1.64	0.16	-1.93	-1.39
b_Age100c	-3.26	1.04	-4.95	-1.49
b_Pclass2:Sexmale	-0.75	0.24	-1.14	-0.36
b_Pclass3:Sexmale	0.86	0.19	0.55	1.19
b_Pclass2:Age100c	-3.38	1.46	-5.81	-0.96
b_Pclass3:Age100c	-0.16	1.22	-2.14	1.88
b_Sexmale:Age100c	-2.45	1.05	-4.19	-0.74
b_Pclass2:Sexmale:Age100c	-0.11	1.47	-2.54	2.37
b_Pclass3:Sexmale:Age100c	1.92	1.19	0.00	3.89

Admittedly, even with all variables centered, interpretation is not so easy. And the logit scale certainty doesn't help. With regard to the coefficients of age, the first thing we observe is the strong negative main coefficient, indicating a strong drop in survival rates for older people. This effect is not constant across passengers groups though. In fact, the it appears to be even stronger (i.e., more negative) for men and second class passengers. For third class passengers

the effect of age is not very different from the main effect, likely because the survival rates of third class passengers were already generally low to begin with. Another thing that becomes apparent is that age coefficients are quite uncertain with posterior standard deviations of over 1 and large credible intervals. All of these aspects can also be found in the corresponding `conditional_effects` plot:

```
conditional_effects(
  fit_titanic4, effects = "Age100c:Pclass",
  conditions = make_conditions(fit_titanic4, "Sex")
)
```



The `effects` argument can only handle two predictors at once, the first being shown on the x-axis, the second being shown as separate color coded lines. Accordingly, if we want to illustrate three predictors together, we need a different mechanism, which comes in the form of the `conditions` argument. This is just a data.frame and for each of its rows, we will get a separate facet in the plot where we conditioned on the predictor values in the corresponding row. You can set up this data.frame manually or, for additional convenience, use `make_conditions`, which will also give you nice facet labels. The data.frame used above looks as follows:

```
make_conditions(fit_titanic4, "Sex")
```

```
     Sex       cond__
1 female Sex = female
2   male   Sex = male
```

Accordingly, it contains all the levels of the `Sex` predictor along with a `cond__` column that will be used as facet labels. If you are specifying your conditions data.frame manually and want to control facet labels, you have to set the `cond__` column yourself. Of course, when we want to use continuous predictors to define the facets, showing all realizations of the predictors is

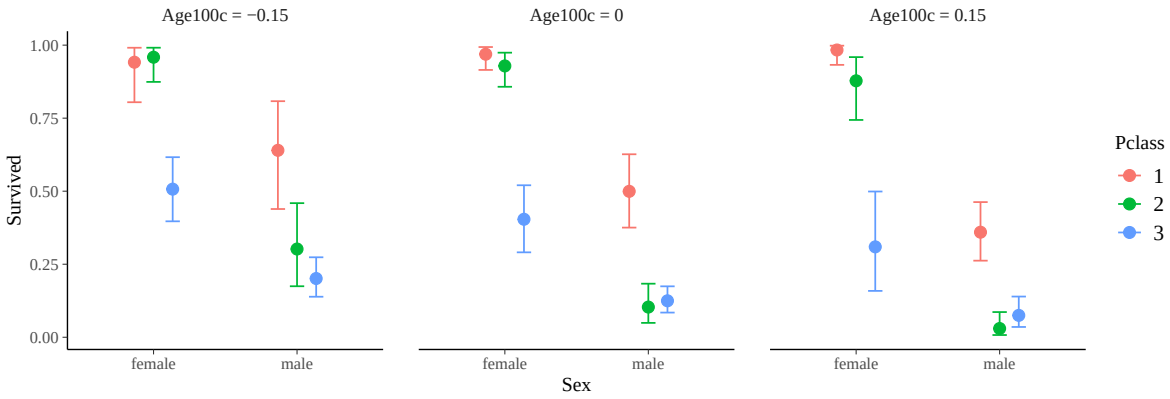
most likely not sensible. Instead, `make_conditons` will create three facets by default, one for the predictor mean and two more of the predictor mean ± the predictor standard deviation:

```
make_conditions(fit_titanic4, "Age100c")
```

```
        Age100c          cond__
1 -1.452650e-01 Age100c = -0.15
2 -2.050084e-17     Age100c = 0
3  1.452650e-01  Age100c = 0.15
```

This reduces information in the plot but at least ensure that it will will not have dozens of facets by default:

```
conditional_effects(
  fit_titanic4, effects = "Sex:Pclass",
  conditions = make_conditions(fit_titanic4, "Age100c")
)
```



At least to me, this plot looks a bit more messy than the first initial one above, but perhaps that's up to taste. My personal recommendation for a mix of continuous and categorical predictors is to show one of the continuous predictors on the x-axis such that we can see the full value range at least for that predictor.

The results above suggest that age is indeed an important variable. But does it also show up in out-of-sample predictive performance? Let's compare the model with and without age via LOO-CV:

```
loo_titanic3 <- loo(fit_titanic3)
loo_titanic4 <- loo(fit_titanic4)
loo_compare(loo_titanic3, loo_titanic4)
```

```
Warning: Found 1 observations with a pareto_k > 0.7 in model 'fit_titanic4'. We
recommend to set 'moment_match = TRUE' in order to perform moment matching for
problematic observations.
```

```
Error : Not all models have the same number of data points.
```

Hmm, that didn't work apparently. A closer inspection reveals that `Age` has some missing values which lead to a removal of the corresponding rows in the data by default (with a warning). So while the full dataset has 891 observations, `fit_titanic4` was only fit with 714 of them. Comparing models on different test datasets would not be fair, hence `loo` complained. We will learn about how to properly deal with missing values in Chapter X. For now, we will have to find a workaround: What if we ask the model without age to only predict the observations with non-missing age values? We first create the corresponding data subset

```
titanic_sub <- titanic %>%
  filter(!is.na(Age))
```

and then feed in this dataset into the `newdata` argument of `loo`:

```
Warning: Found 1 observations with a pareto_k > 0.7 in model 'fit_titanic4'. We
recommend to set 'moment_match = TRUE' in order to perform moment matching for
problematic observations.
```

```
loo_titanic3_sub <- loo(fit_titanic3, newdata = titanic_sub)
loo_compare(loo_titanic3_sub, loo_titanic4)
```

```
             elpd_diff se_diff
fit_titanic4   0.0       0.0
fit_titanic3 -13.6       7.6
```

Now the comparison seems to work and suggests that indeed including age along with its two-way and three-way interactions with class and sex improve predictive performance.

But was this a fair comparison now? Well, it depends on the perspective. If we are attempting to win a Kaggle challenge, and want to choose which of the two models we are submitting as our final model, then this comparison is fair. We would just care about the (out-of-sample) predictive performance and ignoring a subset of the data due to missing values is then "the model's problem". Put differently, we do not care about the probabilistic model itself, but only about its test data predictions based on whatever training data it could use during learning.

But there is also a second perspective. What if we care about the epistemic virtue of age as predictor in the titanic accident? That is, what if we want to judge whether the policy of rescuing young people first was actually implemented? In other words, we would care about the true relevance of age and our two models would be representatives of the two hypothesis "age doesn't matter" and "age does matter" respectively. Then, we really care about the inner workings of the model and the role age plays in it; even if we evaluate the model's performance based on out-of-sample predictions.

In the second perspective, the above model comparison would be biased against the model including age, as it had less training data to learn from and hence smaller potential to predict well just for this reason. Accordingly, to address this epistemic question fairly, we would want to fit both models on the same training data, that is, would need to refit `fit_titanic3` on the `titanic_sub` dataset before running `loo`. I leave this to you as an exercise.

I have to add that there is an ongoing debate on whether cross-validation is at all a valid procedure for the discussed second perspective, but this would lead us too much astray from what we are actually trying to do here. If you are interested in the details check out CITE.

Summarizing the results so far, we may conclude something along the following: Young and/or female people had a substantially higher probability of survival than older men. Unless you were a third class passenger. There, the only people with with a decent chance of survival were young women and girls.

3.4.4 Changing the Link Function

The link function is an essential part also of GLMs for binary data, and more generally for all double bounded responses. brms offers a wide range of options, comprising the most common link functions, including logit (default), probit, cloglog, and cauchit, an illustration of which you can find in Figure X.

At the start of this chapter, I have motivated link functions primarily through pure necessity in order to ensure that predictions remain in the valid (location) parameter range. But for these double-bounded link function applied to binary data – and their generalization – we have access to another interpretation that goes as follows: Assume there is a unbounded latent (i.e., unobservable) variable that gives rise to the binary response through a thresholding process. If the latent variable is smaller than or equal to the threshold, the binary response will be 0 ("no event"). If the latent variable is larger than the threshold, the binary response will be 1 ("event"). Our linear predictor can be thought of as the mean of the latent variable and our threshold can be thought of a being fixed to 0 on the latent scale. As the linear predictor increases, we are more likely to exceed the threshold. Of course, this latent variable is not deterministic but has a distributions, which is implicitly given by the link function. In fact, the response function of a double bounded link is nothing else than the cumulative distribution function (CDF) of the latent variable's distribution.

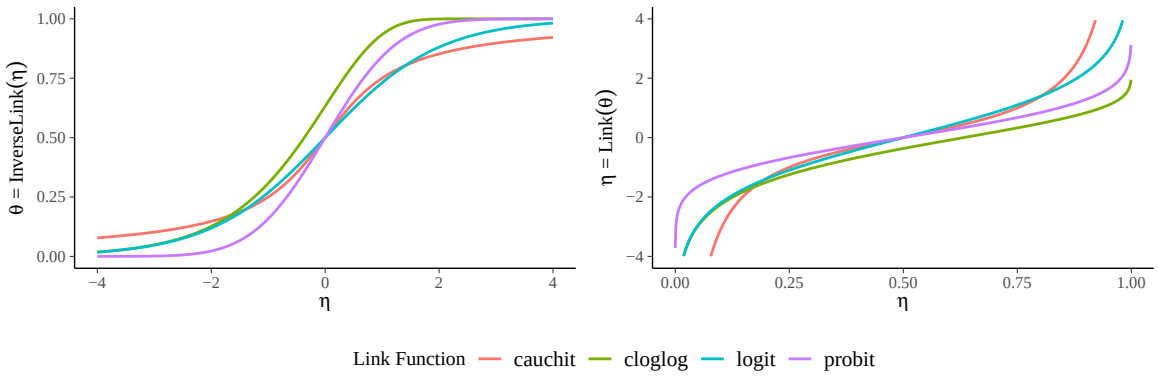


Figure 3.5: Illustration of several common links and corresponding response (inverse link) functions.

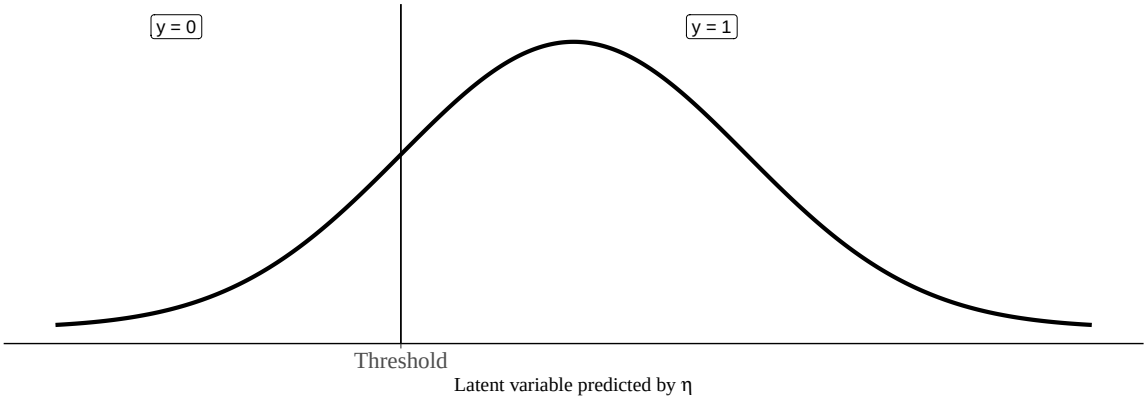


Figure 3.6: Illustration of the thresholding procedure implied by link functions for models of binary data.

The assumption of such a latent process variable is not necessary to justify utilizing these links, but it may at least help with intuition. Sometimes, we can actually argue for the existence of such a latent variable, for example, in some psychological applications: Suppose we ask an individual whether or not they like a particular book, giving them only the option of "I like" or "I don't like" the book. The individual will certainly be able to give a much more nuanced view of the book, all of which will contribute to how much they like the book overall. This "overall liking" may be well understood as an unbounded continuous variable existing in the individual's mind. Of course, we cannot observe this variable directly and our access to it is only indirect by asking the "do you like it or not" question. Accordingly, this question induces a thresholding process in the individual's mind, forcing them to reduce their nuanced opinion to a binary response. We will revisit and extend this perspective in the context of ordinal modeling in Chapter X.

Back to our running example, the survival of passengers on the Titanic may or may not "truly" have an underlying latent variable, but it doesn't matter. We can freely choose our link function in any case. For example, we can use the `cloglog` link, which implies an asymmetric response function with a fatter right tail (see Figure X). That is, even if the odds of survival are stacked against an individual, say, because they are an old men in the 3rd class, they may still be some non-negligible chance of surviving due to random lucky factors. At least a higher chance than a symmetric link function would assign to this individual under the same circumstances.

```
fit_titanic5 <- brm(
  Survived ~ Pclass * Sex * Age100c,
  data = titanic, family = bernoulli("cloglog"),
  prior = prior(normal(0, 3), class = "b")
)
```

Although sampling worked well, a closer inspection of the chains reveals a bunch of warnings that all look like this:

```
Chain 1: Rejecting initial value:
Chain 1:   Log probability evaluates to log(0), i.e. negative infinity.
Chain 1:   Stan can't start sampling from this initial value
```

This indicates the the chain had trouble starting at the randomly chosen initial values. Eventually the chains started so we don't need to bother really. But sometimes, finding good random initial values is so difficult that Stan terminates after (by default) 100 attempts. In fact, this is something we actually see quite often in cloglog models. For example, if we used the unscaled and uncentered age in the formula, i.e., `Survived ~ Pclass * Sex * Age` an reran `fit_titanic5` with it, we would see exactly this problem. In addition or alternative to centering and scaling predictors, you can also change the range where initial values are drawn. If you are using the `rstan` backend to brms (the default), this can be done via the `init_r`

argument. Setting it to values smaller than the default of `2`, say, to `0.1` or so will enforce initial values to be much closer to zero on the unconstrained parameter scale, which often makes it easier to find good initial values. If you are using the `cmdstanr` backend, you can use the `init` argument for the same purpose. In both backends `init = 0` will initialize all parameters at zero on the unconstrained scale. This is not ideal as it implies the same initial values for all chains, which may hamper thorough posterior exploration, but I have often had practical success with this option if all of the previously described options failed.

Back to our actual cloglog model, let's check out the summary output:

```
summary(fit_titanic5)
```

```
 Family: bernoulli
  Links: mu = cloglog
Formula: Survived ~ Pclass * Sex * Age100c
   Data: titanic (Number of observations: 714)

Regression Coefficients:
                      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS
Intercept                -0.54      0.09    -0.73    -0.37 1.00     3175
Pclass2                  -0.12      0.14    -0.41     0.15 1.00     1872
Pclass3                  -0.82      0.12    -1.05    -0.57 1.00     2361
Sexmale                  -1.03      0.09    -1.21    -0.86 1.00     2620
Age100c                  -3.17      0.61    -4.37    -2.01 1.00     3033
Pclass2:Sexmale          -0.59      0.14    -0.87    -0.32 1.00     1830
Pclass3:Sexmale           0.35      0.12     0.12     0.59 1.00     2285
Pclass2:Age100c          -2.55      0.92    -4.40    -0.76 1.00     2092
Pclass3:Age100c           0.28      0.78    -1.27     1.82 1.00     2617
Sexmale:Age100c          -2.16      0.60    -3.34    -0.97 1.00     2977
Pclass2:Sexmale:Age100c  -1.44      0.92    -3.24     0.32 1.00     2371
Pclass3:Sexmale:Age100c   1.44      0.78    -0.02     2.96 1.00     2535
                      Tail_ESS
Intercept                 3252
Pclass2                   2378
Pclass3                   3016
Sexmale                   3014
Age100c                   2860
Pclass2:Sexmale           2446
Pclass3:Sexmale           2693
Pclass2:Age100c           3021
Pclass3:Age100c           2859
Sexmale:Age100c           3160
Pclass2:Sexmale:Age100c   2747
```

```
Pclass3:Sexmale:Age100c     2860
```

Compared to the logit model, the coefficients' posteriors of the cloglog model are a bit smaller (closer to zero): That does not mean that the cloglog model estimates relationships are actually smaller. It is at least partially an artifact of the latent scale differences induced by the different link function: The SD of the latent variable implied the logit model is equal to the SD of the standard logistic distribution, which is $\pi/\sqrt{3} \approx 1.81$. In contrast, the cloglog link implies a standard Gumbel distribution (aka. Type-1 generalized extreme value distribution) on the latent scale with an SD of $\pi/\sqrt{6} \approx 1.28$. The latent scale is not identified by the binary response data, so the SD difference is essentially arbitrary; only the distribution's shape matters.

In an attempt to achieve more comparable estimates, we can take the posterior of the coefficients and divide it by their posterior SD, as exemplified below:

```
post_Sexmale4 <- as.data.frame(fit_titanic4)[["b_Sexmale"]]
post_Sexmale5 <- as.data.frame(fit_titanic5)[["b_Sexmale"]]
post_Sexmale4_scaled <- post_Sexmale4 / sd(post_Sexmale4)
post_Sexmale5_scaled <- post_Sexmale5 / sd(post_Sexmale5)
posterior_summary(cbind(post_Sexmale4_scaled, post_Sexmale5_scaled))
```

```
                       Estimate Est.Error      Q2.5     Q97.5
post_Sexmale4_scaled  -9.960098         1 -12.06231 -8.164240
post_Sexmale5_scaled -11.476886         1 -13.49491 -9.633412
```

So there are in fact differences in the coefficients' posterior, not attributable to the scale differences. For the above example the difference is quite small, but for other coefficients, it actually is much bigger. As an exercise, repeat the above code for some of the interaction coefficients (e.g., `b_Pclass3:Sexmale`). That said, in terms of predictions, the change of link function barely seems to matter Indeed, if we run `pp_check`, `conditional_effects` or `loo`, we see that the models perform very similarly. I don't show this here and leave it to you as an exercise.

3.4.5 Binomial Models

So far, the individual responses where an indicator (0 or 1) whether a person survived the titanic accident. But often, double bounded count data are not just 0 or 1 but $0, 1, 2, \ldots, M$ for a given, an (usually) a priori known M. In fact, we can transform our present data to be in this format. Suppose we are just interested in the predictors `Pclass` and `Sex` as well as their interactions. Then, it is sufficient to just count how many people in a group (say, women in the first class) survived out of all people in that group. To make this more concrete, let's restructure the data this group format.

```
titanic_group <- titanic %>%
  group_by(Pclass, Sex) %>%
  summarise(Survived = sum(Survived), M = n())
```

Pclass	Sex	Survived	M
1	female	91	94
1	male	45	122
2	female	70	76
2	male	17	108
3	female	72	144
3	male	47	347

As we will see, this data set formally containing just 6(!) rows is sufficient to estimate the main effects and interactions of class and sex, which together implies 6 regression coefficients. Since we assumed each individual survival indicator to be Bernoulli distributed with survival probability θ, it follows that the sum of M such indicators is binomial distributed with the same θ and number of trials M (here number of people in a group). The term "trials" comes from situations where we are repeating an experiment independently multiple times, say, we throw the same coin M times and count how often the coin lands on head. Of course, in our titanic example the indicator we are measuring is not really a "trial" but an individual human but the statistical model neither knows nor cares whether the data we feed it with are coin throws or indicators of individual humans' survival. We as data analysts are the ones who need to care. Okay, let's fit our binomial model at last:

```
fit_titanic_binom1 <- brm(Survived | trials(M) ~ Pclass * Sex,
                          data = titanic_group, family = binomial("logit"))
```

Sampling works well and fast, with excellent convergence. Apparently, the model did not bother that the number of "observations" (rows) in the data is the same as the number of regression coefficients. If we tried the same on 6 rows of Bernoulli (binary) data, the model will fail miserably because the amount of information contained there is way smaller than the amount of information contained in a data point comprising the observations of all individuals in a group.

When investigating the summary output of the binomial model, we see that its results are the same as the corresponding Bernoulli model `fit_titanic1`, up to MCMC error. This is no coincidence. We can either specify our binomial model on the level of the individual "trials" or on the level of groups. But sometimes grouping isn't possible, at least now without loss of information. For example, if we wanted to include `Age` as additional predictor in the binomial model, without loss of information, we would need to create groups per year of age. The resulting reduction in data rows would not be that big. We can see this by computing

```
titanic_group2 <- titanic %>%
  group_by(Pclass, Sex, Age) %>%
  summarise(Survived = sum(Survived), M = n())
```

where we find `titanic_group2` to have `nrow(titanic_group2)` rows, compared to `nrow(titanic)` rows in the original dataset. Of course, we could choose to compress age into groups, say, from 0 to 10, 11 to 20, etc. but this would result in a loss of information in the age predictor that we should seek to avoid. As an exercise, you may want to fit binomial model to the clustered age data (each cluster having an age range of 5, 10, or 20 years) and see how the posterior changes in comparison to the models without such clustering.

3.5 Summary

We have seen that GLMs can be easily specified in brms by changing the `family` argument, while keeping the model formula the same (or only adding one term as in binomial models). The likelihood families we have used in the examples here are only a small subset of all the families implemented in brms. We will see many of them in use in the following chapters. If you want to get an overview right now, check of the `?brmsfamily` help page. Regardless of the GLM you choose, the principles that we learned here stay the same: We are changing the family argument, and perhaps the link function if We don't want to use the default. And then you are ready to go. The price we pay is a more challenging interpretation of the regression coefficients but we can help with that with some nice and quick predictive visualizations. In many of the upcoming chapters, we will use GLMs as the basis for more complicated models.

4 Linear multilevel models

4.1 Setup

```
library(ggplot2)
library(patchwork)
library(bayesplot)
library(brms)
```

4.2 Introduction

Most real-world datasets are not just a collection of independent observations measured cleanly in an experimental setting with only one observation per individual. Instead, we are more often than not confronted with complex dependency structures, for example, because of natural groupings of individuals or repeated measurement of the same individuals. Here, "individuals" can mean many different things, for example, humans, animals, cells, classes, laboratories, studies, etc. In a longitudinal study in psychology, for example, the individuals will be humans measured multiple times over the course of the study. Naturally, the observations belonging to the same individual are highly likely to be dependent on each other, as they share the same source, namely said individual, even if they had been gathered under different experimental conditions. Also, some individuals are often more similar than others, as the result of groups occurring in the data. For example, students in the same class, are highly likely dependent as they share the same class rooms, peers, teachers etc. We can of course choose to ignore these dependencies in our modeling, but then pay the price in the form of overconfident and potentially biased inference, as we will see later on in this chapter. Multilevel models, on the other hand, are built to express such dependencies explicitely, thus helping to obtain trustworthy inference from complex-structured (multilevel) datasets. Due to the frequency with which datasets contain multilevel structure, I consider multilevel models as the *default* model class for statistical modeling, Bayesian or otherwise.

As a running example, we will use the `sleepstudy` dataset originally published by Belenky et al. (2003), and available in R from the `lme4` package (Bates et al. 2015). It contains the average reaction times per day across 10 consecutive days for a selection 18 subjects in a sleep deprivation study. After the first two days of adaptation and training, sleep deprivation

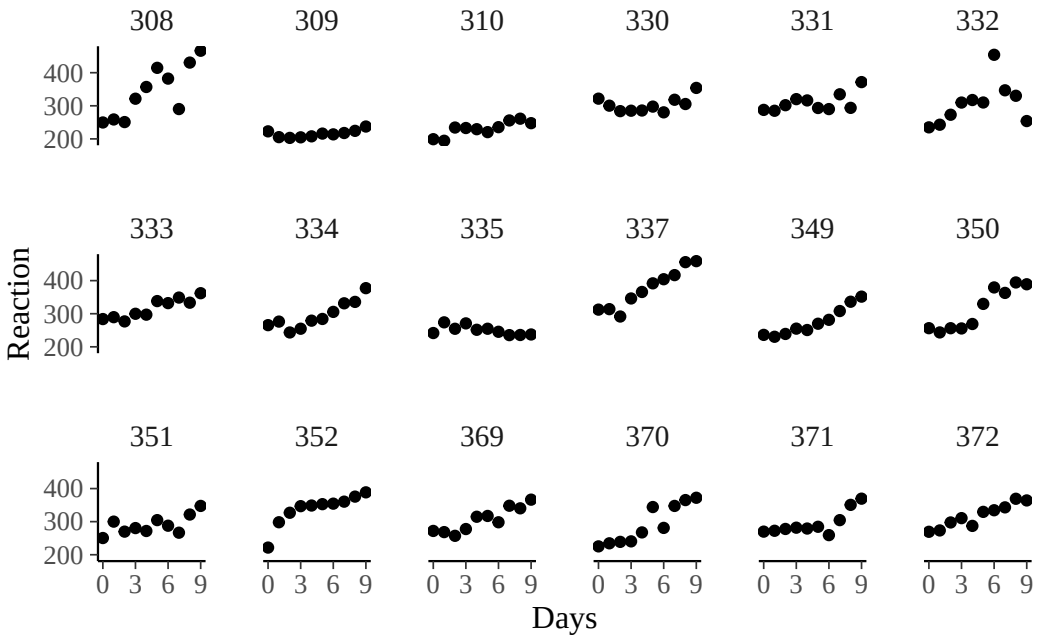
started after day 2, which meant that subjects were allowed to sleep for only a maximum of 3 hours per night. If you have ever slept this little for several nights in a row (I definitely have), you will know that this messes with your concentration so a lot of tasks become slower and harder overall. It is not hard to hypothesize that several days of consecutive sleep deprivation will lead to slower reaction times overall, the details of which we will analyse below. Let's load the data first and get an initial overview:

```
data("sleepstudy", package = "lme4")
head(sleepstudy)
```

```
  Reaction Days Subject
1 249.5600    0     308
2 258.7047    1     308
3 250.8006    2     308
4 321.4398    3     308
5 356.8519    4     308
6 414.6901    5     308
```

The datasets is very simple in structure, containing only 3 variables: `Reaction` contains the average reaction time (in milliseconds) for a specific `Day` (0 to 9) and a specific `Subject`. In total, we have access to 180 observations, 10 day for each of 18 subjects. Clearly, it contains multilevel structure in the form of repeated measurements of the same subjects. In other words, individual observations are nested within subjects. Let's investigate the data graphically:

```
sleepstudy %>%
  ggplot(aes(Days, Reaction)) +
  geom_point() +
  facet_wrap("Subject", nrow = 3) +
  scale_x_continuous(breaks = c(0, 3, 6, 9))
```



When looking at the plot, we see that we may not need any modeling to understand that the subjects' reaction times increase over the days of consecutive sleep deprivation. Moreover, we see that the performance varies quite drastically between subjects, both in the initial baseline reaction time at Day 0 and in how much reaction times increase over the days. What we don't see immediately are the results averaged across subjects: In many real-world analysis scenarios, we are not interested in the specific behavior of specific subjects, but rather in what we learn over the population of individuals more generally. For example, we might ask about the average reaction time at baseline as well as the average change of reaction time over the days. In other words, we want to learn something *general* (average) about how humans react to sleep deprivation. What is more, we would like to understand how much individuals tend to *vary* around those average results. And if that wasn't enough, we would like to obtain trustworthy *uncertainty estimates* of both the average results and their variations across the population of individuals. Bayesian multilevel models can provide us with answers to all of these three questions. How good those answers are depends, as always, on how well our modeling assumptions align with the data being analysed.

4.3 Complete pooling

As our first model, we will, for the moment, throw all of these considerations of multilevel structures out of the window and just fit a basic linear models, with just an average intercept and slope:

$$y_n \sim \text{normal}(b_0 + b_1 x_n, \sigma)$$

We call this a *complete pooling* model as we ignore potential variations across individuals and thus completely pool all their data together to only from a single set of average coefficients; as if all individuals had the same intercept and the same slope.

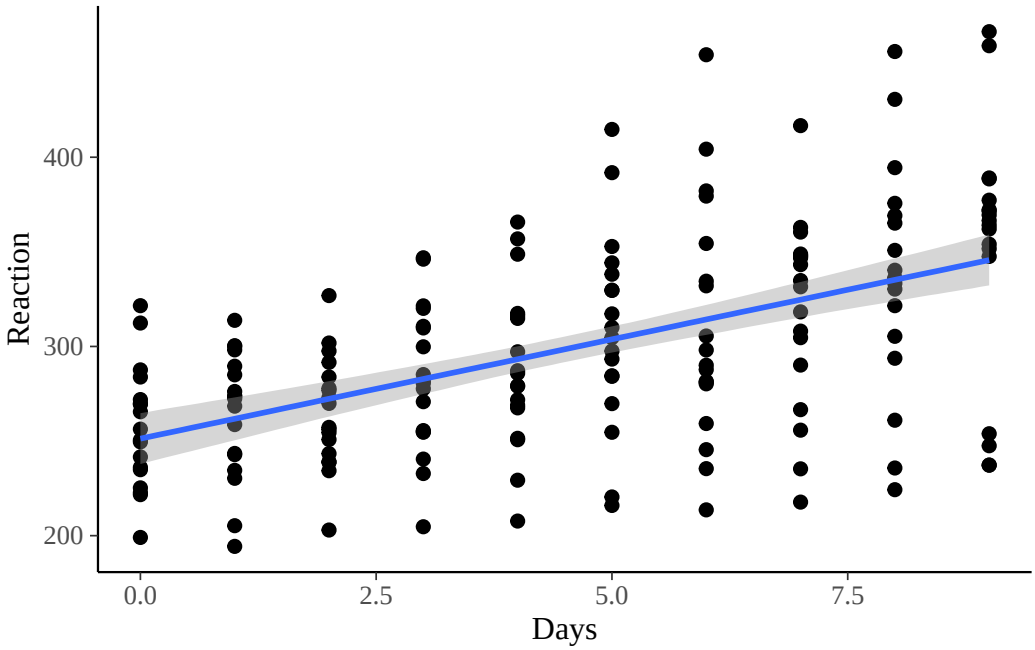
```
fit_sleep1 <- brm(Reaction ~ 1 + Days, data = sleepstudy)
```

```
summary(fit_sleep1)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Reaction ~ 1 + Days
   Data: sleepstudy (Number of observations: 180)

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept   251.38      6.78   237.86   264.72 1.00     3934     2668
Days         10.47      1.30     7.93    13.04 1.00     3942     2734

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    48.02      2.61    43.37    53.59 1.00     3982     2628
```

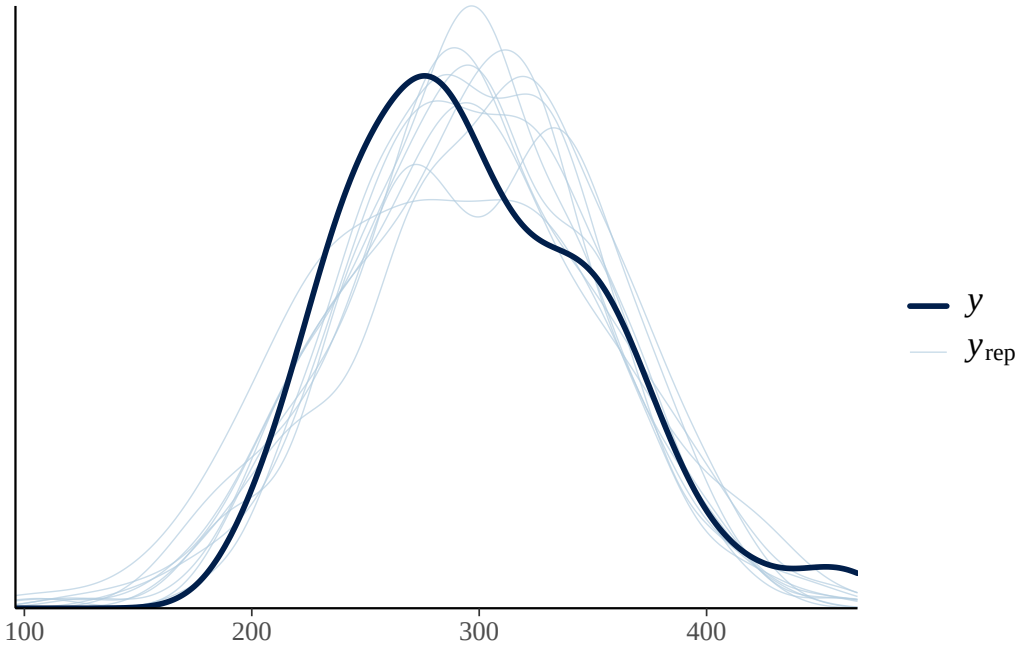
We learn from the results that, on average, the model estimates a baseline of around 250 milliseconds and a slope of about 10.5 additional milliseconds per day, with a 95% credible interval of 8 to 13 milliseconds. Visually, the predictions look as follows:

```
plot(conditional_effects(fit_sleep1), points = TRUE)
```



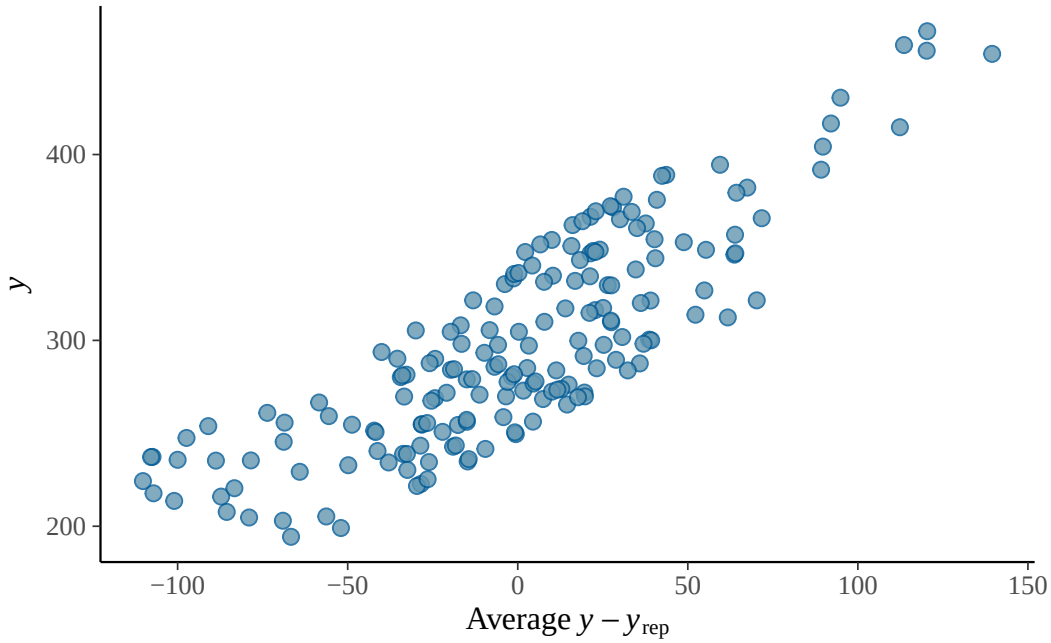
Clearly, average reaction times increase over the course of the 10 days, and the model is quite confident by how much reaction times are expected to increase. In this simple example, we are using just a single predictor (`Days`), so the comparison of model-based predictions and observations in this plot can be used as sensible indication of model fit. We see that the variances of observations at the early days of the study are much smaller than the variances towards the end of the study. Let's see what more standard posterior predictive plots tell us about the model:

```
pp_check(fit_sleep1, type = "dens_overlay")
```



The default predictive check with `type = "dens_overlay"` is painting a picture of a supposedly reasonably fitting model. At least there are no huge indications that the model gets the marginal distribution of the response variable wrong. If you have read the previous chapters carefully, you will already know to not trust this result too much. So what do we learn from a plot of predictive errors?

```
pp_check(fit_sleep1, type = "error_scatter_avg")
```



That doesn't look so good anymore. Remember that, for a reasonable fitting model, we would expect to see a big point cloud in such a plot, without any clearly visible patterns. Any deviation from that points to some potential misfit. And indeed, there is a strong linear pattern in this plot. Specifically, large negative errors (lower observed than predicted responses) are associated with small observed responses. Conversely, large positive errors (higher observed than predicted responses) are associated with large observed responses. Accordingly, there is apparently some strong signal in the data that our complete pooling model fails to account for. Based on our discussion at the start of this chapter, you should already have a pretty good hunch on which that is. So let's dive right in and starting fitting multilevel models.

4.4 Partial pooling: Varying intercepts

As a first step, we will assume the intercept b_0 to vary across subjects, by replacing b_0 with a set of intercepts b_{0j}, one for each subject j:

$$y_n \sim \text{normal}(b_{0j[n]} + b_1 x_n, \sigma)$$

I use the notation $j[n]$ to indicate the group j to whom observation n belongs. This likelihood alone does not yet specify a multilevel model yet. In fact, it is just a linear model with cell mean coding on the subject factor. The magic begins as soon as we set a special kind of prior on the varying intercepts:

$$b_{0j} \sim \text{normal}(b_0, \sigma_0)$$

If we were to fix the mean b_0 and the standard deviation σ_0 to some values, this would just be a boring normal prior that we have seen time and again already in previous chapters. However, in multilevel models, we will not only assume the b_{0j} but also b_0 and σ_0 to be parameters, estimated from the data. In other words, we have a hierarchy of parameters, where the prior on the lower level parameters b_{0j} is determined by the high-level (hyper-)parameters b_0 and σ_0. To complete the model, we now also have to set priors on b_0, σ_0, and the residual standard deviation σ, but we do not focus on these priors for now. It is really the hierarchical prior above that sets multilevel models apart from standard regression models.

First, let us talk about what assumptions we convey with the hierarchical prior. Since every subject's intercept b_{0j} has the same prior normal(b_0, σ_0), we a priori do not provide any information which of the subjects may have a smaller and which may have a larger intercept. We do of course assume there will be variation between subjects (unless $\sigma_0 = 0$), but this assumption is symmetric as it treats all subjects equally. Does that mean we assume subjects to be *independent*? Not quite, since all subjects share the same hyperparameters. If, for example, the overall intercept b_0 increases, the subjects' individual intercepts will tend to increase too. Thus, speaking more generally, by means of the hierarchical prior with estimated hyperparameters, we induce a positive correlation between the lower-level parameters. So the intercepts are in fact dependent, but in a symmetric manner. We call this assumption *exchangeability*. Mathematically speaking, their joint prior is the same regardless of the subjects' ordering. For example, we could reverse the ordering of the subjects' indices and the hierarchical prior would still be the same:

$$p(b_{01}, b_{02}, \dots, b_{0j-1}, b_{0j}) = p(b_{0j}, b_{0j-1}, \dots, b_{02}, b_{01})$$

This in intuitively clear when we look at the hierarchical prior specification above, but is still crucial to point out so we understand when the assumptions of multilevel models are justified, and when they are not. But what do we actually achieve by means of this prior and its exchangeability assumption?

The key lies in understanding the difference between independence and exchangeability. If we had assumed the intercepts to be mutually independent, as is done by a standard regression model, the intercepts could not share information with one another. In other words, if some intercepts were large that would have no influence on the other intercepts, as least not as conveyed through the prior. Now, with our hierarchical prior, if some intercepts are large, this will increase the overall intercept b_0, which in turn influences the prior of all intercepts, pushing all intercepts up. Of course, this exchange of information is symmetric across subjects, so each intercept influences all other intercepts via its influence on the hyperparameters. We can also say it like this: Through the hierarchical prior, all lower-level parameters are shrunken towards their hierarchical mean parameter. This sharing of information is what we call *partial*

pooling following Gelman and Hill (2006). The degree of shrinkage induced by partial pooling is influenced (a) by the distance between the individual parameters and the hierarchical mean (higher distance implies higher shrinkage) and (b) the size of the hierarchical variations parameters (larger variation implies smaller shrinkage). In our example, if the intercept of a specific subject is far away from the hierarchical mean intercept b_0, it will be shrunken more than those subjects' intercepts closer to b_0. But the degree of that shrinkage will be modulated by the size of the hierarchical standard deviation σ_0.

Is *partial pooling* something we actually we want? I would argue that it is, indeed, desirable at least is many cases. Thinking about the sleepstudy example, all subjects are healthy, adult humans so they inevitably share some similarities within one another. For example, if most subjects have an average response time of around, say, 300 ms for the task at hand, it becomes quite unlikely that some subjects will have an average response time of, say, 10 seconds. In other words, by knowing something about some subjects, we also know something about the other subjects. That is not else than what the partial pooling property induced by the hierarchical prior attempts to express.

Before we start fitting multilevel models in brms, we need to make on further change in the model. Nothing that actually changes it's meaning. Just a reformulation that makes model specification more compatible with R formula syntax. Instead of directly modeling the b_{0j} as centered around b_0, we define a new set of intercepts $\tilde{b}_{0j}$, which we model as centered around 0 instead:

$$\tilde{b}_{0j} \sim \text{normal}(0, \sigma_0)$$

Now that b_0 no longer appears in the hierarchical prior, we add it directly to the likelihood:

$$y_n \sim \text{normal}(b_0 + \tilde{b}_{0j[n]} + b_1 x_n, \sigma)$$

The two formulations are equivalent since $b_{0j} = b_0 + \tilde{b}_{0j}$ so the model assumptions remain the same. That said, the second formulation we just introduced has the advantage that all its terms, the overall intercept b_0, the overall slope of b_1 of predictor x, and the varying intercept $\tilde{b}_{0j}$ all appear in the same place, namely the mean parameter of the likelihood. It is now easy to express this mean parameter as an R formula.

Inspired by the syntax of the `lme4` package, multilevel terms are specified in brms as `(terms | group)`. Accordingly, if we want to add a varying intercept over subjects, we have to add `(1 | Subject)` to the model formula. Together with the overall intercept and the overall slope of `Days`, our varying intercept model formula becomes `Reaction ~ 1 + Days + (1 | Subject)`. The order of the terms doesn't matter so we can also write, for example, `Reaction ~ 1 + (1 | Subject) + Days`, but it is more common to add multilevel terms at the end. With this formula in hand, and for now trusting the default priors, we are ready to fit our first multilevel models in brms:

```r
fit_sleep2 <- brm(Reaction ~ 1 + Days + (1 | Subject),
                  data = sleepstudy)
```

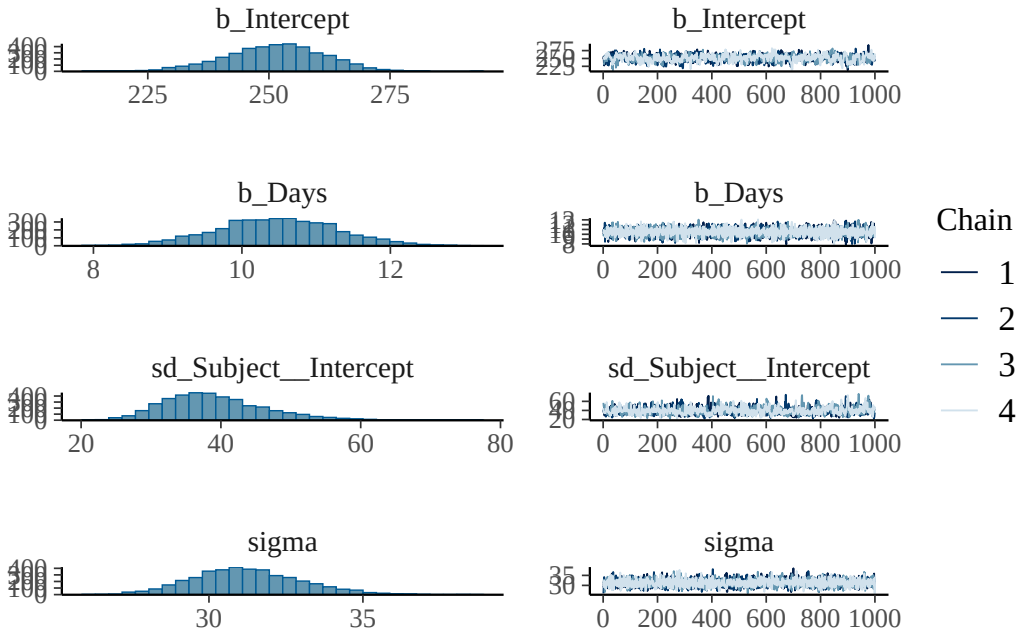
```r
summary(fit_sleep2)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Reaction ~ 1 + Days + (1 | Subject)
   Data: sleepstudy (Number of observations: 180)

Multilevel Hyperparameters:
~Subject (Number of levels: 18)
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)    39.25      7.55    27.18    56.75 1.00      824     1307

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept   251.26     10.20   230.47   270.01 1.00      704     1486
Days         10.47      0.82     8.87    12.07 1.00     4072     2734

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    31.24      1.76    28.01    34.88 1.00     3268     3246
```

In the summary output, notice the new part named "Multilevel Hyperparameters", with the single row termed `sd(Intercept)` summarizing the standard deviation σ_0 of the intercepts over subjects. We can, of course, also understand the overall intercept `b_0` as a hyperparameter, but since this parameter is also present for non-multilevel models we just leave it under "Regression Coefficients". Based on model results, σ_0 has a posterior mean of around 39 ms with a 95% credible interval ranging between 27 and 57. Judged by these credible interval bounds relative to the mean, we see that the posterior of σ_0 is somewhat right skewed since the distance of the mean to the upper bound is larger than its distance to the lower bound. When combining the estimates of σ_0 with the posterior mean of $b_0 \approx 250$ ms, we can infer that most subjects have a baseline reaction time of roughly 150 to 350 ms.

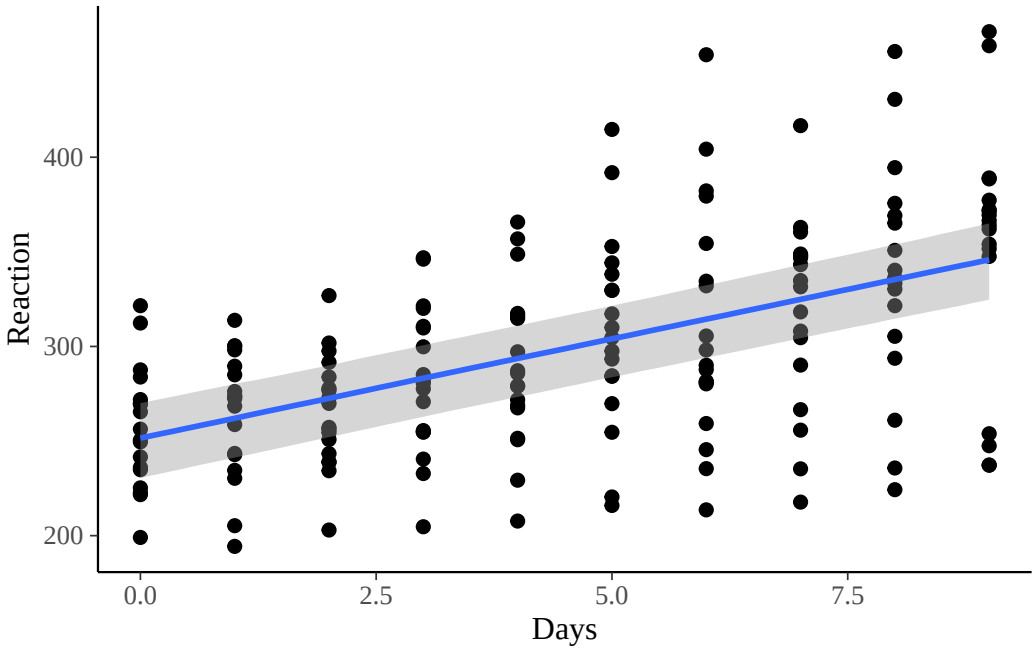
The summary output does not show summaries of the individual intercepts for the subjects because that would quickly clutter the output for more than a handful of grouping levels. Similarly to the `summary` method, also the `plot` method will not show any varying coefficients by default.

```
plot(fit_sleep2)
```



In the plot, we nicely see the slightly right skewed posterior distribution of σ_0 ranging roughly between 25 and 60 ms, confirming graphically what we have seen numerically in the summary output. We can also use `conditional_effects` as usual to visualize the mean predictions associated with the covariates:

```
plot(conditional_effects(fit_sleep2), points = TRUE)
```



By default, `conditional_effects` will ignore all varying coefficients, that is, assume them to be 0. As a result, the prediction line we are seeing is the posterior of $b_0 + b_1 x$, just like it was for the basic linear model without any multilevel terms. Yet, the predictions clearly look different than that of the basic linear model, in particular more uncertain. What happened is that, by telling the model about the different groups and their varying intercepts, the overall intercept b_0 became more uncertain since it is now only informed by 18 "data points", the 18 varying intercepts that are also uncertain themselves, rather than by all the 180 actual observations.

In most real-world applications of multilevel models that I came across, people do not actually care about the varying coefficients. We just need to model them to appropriately to better calibrate the uncertainty of the overall coefficients. That said, we can of course investigate the varying coefficients using several methods dedicated for this purpose. The `ranef` method will return the varying coefficients as estimated by Stan, that is, those $\tilde{b}_{0j}$ centered around zero:

```
ranef(fit_sleep2)
```

```
$Subject
, , Intercept

      Estimate Est.Error        Q2.5      Q97.5
308  40.764334  13.05198   15.899127  66.418306
309 -77.673390  13.18974 -104.200083 -51.885567
```

```
310 -62.807936  13.12826  -87.931626 -36.913359
330   4.727497  13.18675  -20.656765  31.008703
331  10.313182  12.76427  -14.052145  35.409497
332   8.532565  13.50424  -17.246923  35.750218
333  16.461085  13.25355  -10.564029  42.868052
334  -2.693324  12.87025  -27.420545  23.813087
335 -45.102319  13.17972  -70.871325 -18.414603
337  72.440768  13.35906   46.624679 100.015076
349 -20.982987  13.00369  -45.760081   5.331026
350  14.327827  13.25005  -11.406318  40.228837
351  -7.744090  13.07795  -32.987663  18.280651
352  36.556072  13.24638   11.400835  63.582834
369   7.100018  13.24155  -17.787281  33.204618
370  -6.288820  13.18623  -31.890199  19.358234
371  -3.014845  13.46821  -28.991997  23.702356
372  18.209226  13.21784   -7.702734  44.702130
```

For example, we see that Subject 308 is around 40 ms slower at baseline than the mean across subjects, while Subject 309 is around 78 ms faster at baseline than said mean. We also see that all these intercept estimates have considerable uncertainty with 95% credible intervals spanning around 50 ms. This is not surprising if we consider that each subject has only 10 associated data points. Even though the partial pooling property allows to borrow information also from the other subjects, the combine information is still relatively sparse leading to the large uncertainty estimates.

While the $\tilde{b}_{0j}$ coefficients are the ones being estimated by Stan, it is usually the "actual" varying coefficients $b_{0j} = b_0 + \tilde{b}_{0j}$ that we are care about. We could go ahead and extract the posterior draws of b_0 and $\tilde{b}_{0j}$ and than add them manually as we have learned in the past chapters, but since this is such a common task in multilevel models, brms has the `coef` method readily made just for this purpose:

```
coef(fit_sleep2)
```

```
$Subject
, , Intercept

    Estimate Est.Error     Q2.5    Q97.5
308 292.0227 10.350208 271.9308 311.9030
309 173.5850 10.375604 153.0521 193.8731
310 188.4504 10.403551 167.9796 209.3581
330 255.9859 10.197027 236.4471 275.7250
331 261.5716 10.004456 241.7466 280.8069
```

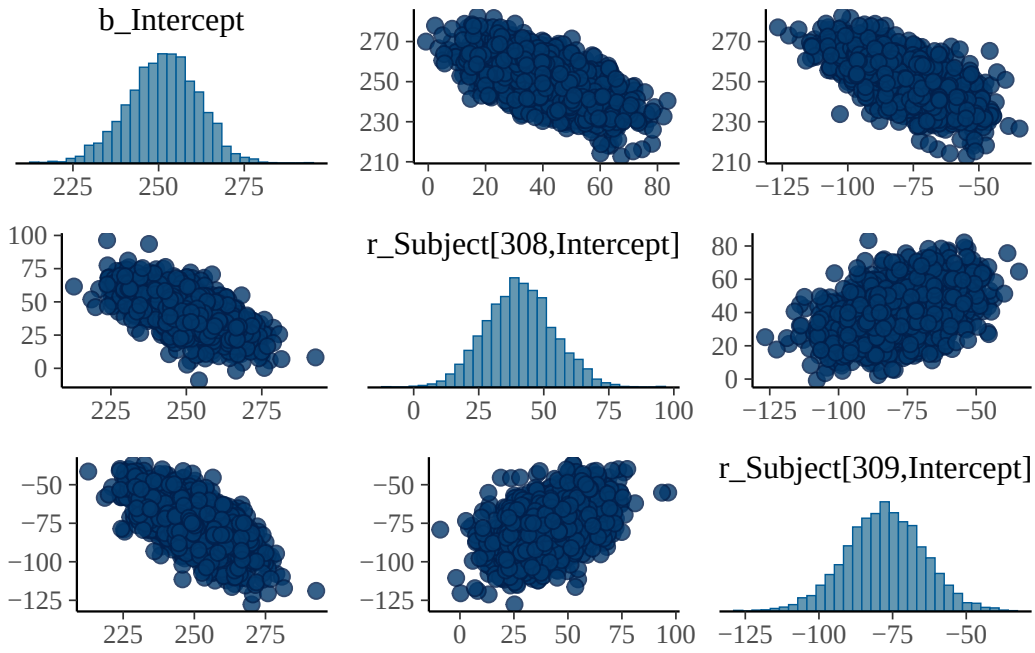
```
332 259.7909 10.466199 239.5445 280.4010
333 267.7195 10.524726 247.0892 288.6462
334 248.5651 10.052758 228.5655 268.3304
335 206.1561 10.364296 185.9868 226.3377
337 323.6991 10.686943 302.5733 343.9014
349 230.2754 10.138772 210.5526 250.1838
350 265.5862 10.362104 245.2435 285.7808
351 243.5143  9.990616 223.7420 263.2507
352 287.8144 10.369607 267.1215 308.1426
369 258.3584 10.418051 238.3001 278.3517
370 244.9696 10.332392 224.9583 264.9751
371 248.2435 10.679777 226.8657 269.4435
372 269.4676 10.472368 248.8821 289.5460

, , Days

    Estimate Est.Error     Q2.5    Q97.5
308 10.47042 0.8150733 8.874827 12.06604
309 10.47042 0.8150733 8.874827 12.06604
310 10.47042 0.8150733 8.874827 12.06604
330 10.47042 0.8150733 8.874827 12.06604
331 10.47042 0.8150733 8.874827 12.06604
332 10.47042 0.8150733 8.874827 12.06604
333 10.47042 0.8150733 8.874827 12.06604
334 10.47042 0.8150733 8.874827 12.06604
335 10.47042 0.8150733 8.874827 12.06604
337 10.47042 0.8150733 8.874827 12.06604
349 10.47042 0.8150733 8.874827 12.06604
350 10.47042 0.8150733 8.874827 12.06604
351 10.47042 0.8150733 8.874827 12.06604
352 10.47042 0.8150733 8.874827 12.06604
369 10.47042 0.8150733 8.874827 12.06604
370 10.47042 0.8150733 8.874827 12.06604
371 10.47042 0.8150733 8.874827 12.06604
372 10.47042 0.8150733 8.874827 12.06604
```

The output of `coef` follows the same structure as that of `ranef`. However, `coef` shows all coefficients even those without varying coefficients associated with it. As a result, we also see per-subject coeffcients of `Days`, which are however constant across subjects and just resemble the overall coefficient that we alreay saw in the summary output.

When we compare the results of `coef` and `ranef`, we see that not only `coef` shows estimates around b_0 instead of around 0 but also that the uncertainty of `coef` is actually *smaller* than that

of `ranef` as can be seen by both the posterior standard deviations (column `"Est.Error"`) and the credible interval range. This is curious since, in `coef`, we actually have added something, namely (the posterior of) b_0, which should intuitively lead to an increase in uncertainty rather than a decrease. To understand what has happened, let's look at the posterior of b_0 as well as the b_{0j} for the first two subjects via a posterior pairs plot:

```
vars <- c("b_Intercept",  "r_Subject[308,Intercept]", "r_Subject[309,Intercept]")
pairs(fit_sleep2, variable = vars)
```



We see a strong negative correlation between b_0 on the one hand and the b_{0j} on the other hand. Conversely, the b_{0j} are positively correlated with one another, as we also see from computing the correlations explicitly:

```
draws_sleep2 <- as.matrix(fit_sleep2, variable = vars)
colnames(draws_sleep2) <- c("b0", "b0_308", "b0_309")
cors_sleep2 <- cor(draws_sleep2)
print(cors_sleep2, digits = 2)
```

```
          b0 b0_308 b0_309
b0      1.00  -0.63  -0.63
b0_308 -0.63   1.00   0.45
b0_309 -0.63   0.45   1.00
```

This happens because, in the likelihood, we add b_0 and b_{0j} together such that the both "fight" for the same information in the data, namely the baseline reaction times. As b_0 increases, so must the b_{0j} decrease (and vice versa) in order to properly fit the empirical baseline reaction times. Since all of the b_{0j} fight with b_0 but not with each other, since they relate to data from different subjects, they end up being positively correlated with each other. These correlations are largely not due to any partial pooling effect but simply an artifact of how we have parameterized our model. But what happens if we look at the "actual" varying coefficients obtained via `coef`?

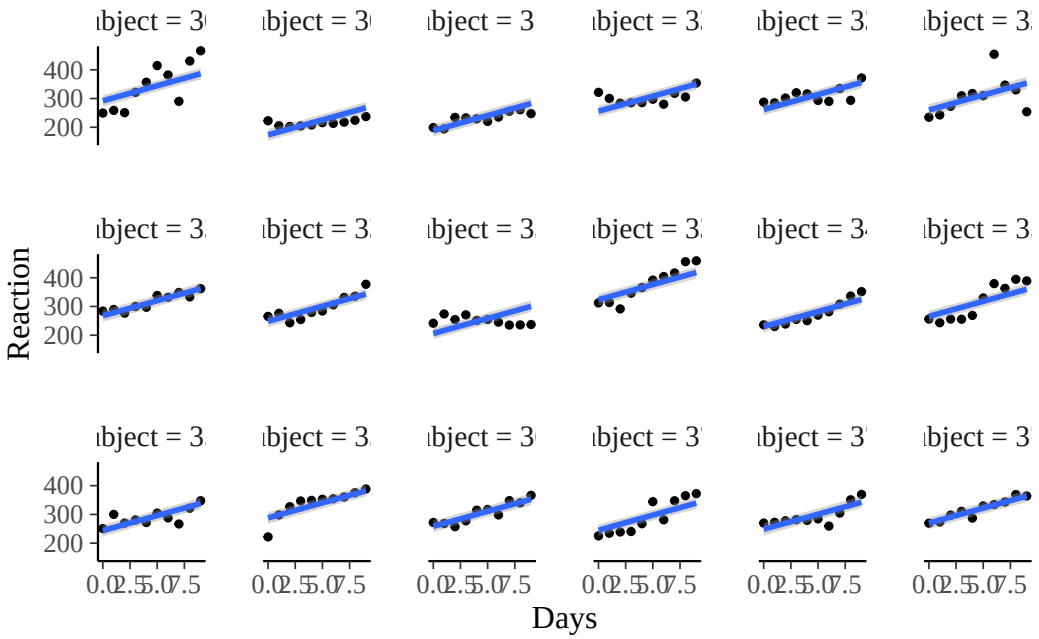
```
coef_sleep2 <- coef(fit_sleep2, summary = FALSE)
print(cor(coef_sleep2$Subject[, 1:5, "Intercept"]), digits = 2)
```

```
     308   309  310  330   331
308 1.00 0.115 0.13 0.10 0.139
309 0.11 1.000 0.17 0.15 0.073
310 0.13 0.166 1.00 0.10 0.110
330 0.10 0.148 0.10 1.00 0.132
331 0.14 0.073 0.11 0.13 1.000
```

As we can see exemplarily for the first five subjects, their varying intercepts are indeed all mildly correlated with each other with a correlation of roughly $r \approx 0.1$. These correlations are now actually due to partial pooling and it shows how the multilevel model enables sharing of information across coefficients. While $r \approx 0.1$ doesn't look like much, remember that it applies to all pairs of varying intercept, such that their *joint* information exhange adds up. For example, when we take the posterior of an individual subject's intercept and correlate it with the posterior of the mean of all other subjects' intercepts, we see a correlation of about $r \approx 0.3$, which is no longer that small. I suggest you to compute this correlation yourself as an exercise. These posterior correlations are not the only way in which hierarchical priors find their way into the posterior and we will come back to this later on.

Let us move on to investigating the per-subject predictions by including the varying intercepts into the prediction plots created via `conditional_effects`:

```
conditions <- make_conditions(sleepstudy, "Subject")
ce <- conditional_effects(
  fit_sleep2, conditions = conditions,
  re_formula = NULL
)
plot(ce, ncol = 6, points = TRUE)
```



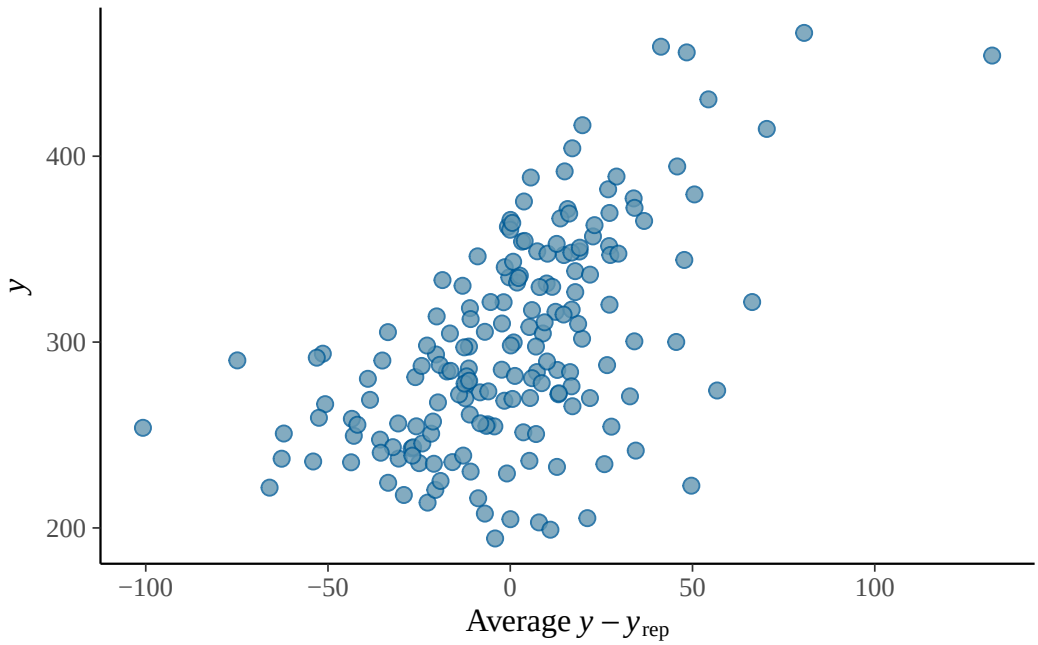
With the `conditions` argument, we indicate that we want to generate one plot per row of the supplied dataframe, here one row per subject. By setting `re_formula = NULL` we indicate that we want the varying coefficients are included in the predictions. By default, `conditional_effects` would ignore them, as we have seen above. The model provides quite reasonable predictions for all subjects, but obviously fails to account for the differences in the slopes. That is, for some subjects with more extreme (large or small) slopes, there is an easily visible difference in the data patterns and the model's predictions. The fact that the varying intercept model fits better than our complete pooling model is also confirmed by the LOO-CV results, which show an ELPD difference of about 68, with a standard error of about 13, in favor of the varying intercept model:

```
loo_sleep1 <- loo(fit_sleep1)
loo_sleep2 <- loo(fit_sleep2)
loo_compare(loo_sleep1, loo_sleep2)
```

```
           elpd_diff se_diff
fit_sleep2   0.0       0.0
fit_sleep1 -68.2      12.5
```

The posterior predictive checks also look better than those of the complete pooling model, but still show a clearly visible relationship between residuals and responses, pointing to some signal in the data that we have failed to model still.

```
pp_check(fit_sleep2, "error_scatter_avg")
```



So far, we have not talked about priors for the hyperparameter σ_0 yet, and instead just used brms' default priors. So what was the default prior actually?

```
prior_summary(fit_sleep2)
```

```
                   prior     class      coef   group lb ub       source
                  (flat)         b                                default
                  (flat)         b      Days               (vectorized)
 student_t(3, 288.7, 59.3) Intercept                            default
     student_t(3, 0, 59.3)        sd                   0         default
     student_t(3, 0, 59.3)        sd           Subject  0    (vectorized)
     student_t(3, 0, 59.3)        sd Intercept Subject  0    (vectorized)
     student_t(3, 0, 59.3)     sigma                   0         default
```

As we can see from the `prior_summary` output, the default prior on standard deviations of the varying coefficients (class `"sd"`) for this model is `student_t(3, 0, 59.3)`; the same as the default prior of the residual standard deviation (class `"sigma"`). Remember from earlier chapters that, internally, such priors for standard deviations are truncated at zero such that we are looking at a half-Student-t prior with small degrees of freedeom (heavy right tails) and a data-dependent scale, that is for linear models equal to the standard deviation of the response

variable. As is the case for many brms' default priors, I don't cliam it is necessarily the best default prior on standard deviation parameters. However, it fullfills two important critiera: (a) It is highly unlikely to be more than weakly informative independent of the specific dataset or model due to the small degrees of freedom and data-adaptivity. (b) It often improves sampling speed and convergence compared to completely flat priors. Later on, in Section 4.10, we will discuss an example on how to specify informative priors for standard deviations of varying coefficients. For now, we move on directly to more complex multilevel models.

4.5 Partial pooling: Varying intercepts and slopes

Above, we have seen that only letting the intercept vary across subjects was insufficient since, apparently, also the effect of the days of sleep deprevation varies across subjects. So let's model this appropriately now. Our likelihood continues to be normal but now the its mean parameter is obtained from both a varying intercept b_{0j} and a varying slope b_{1j} of the respective subject to which the nth observation belongs:

$$y_n \sim \text{normal}(b_{0j[n]} + b_{1j[n]}x_n, \sigma)$$

So far so simple. But now we have to figure out how to write down an appropriate multilevel prior for the pair (b_{0j}, b_{1j}). The most immediate solution is specify two independent multilevel priors, one for each kind of coefficient. This would then look as follows:

$$b_{0j} \sim \text{normal}(b_0, \sigma_0)$$

$$b_{1j} \sim \text{normal}(b_1, \sigma_1)$$

where we now have four hyperparameters, namely, the overall intercept b_0, overall slope b_1 as well as the standard deviations σ_0 and σ_1 for the varying intercepts and slopes, respectively. That is a valid and sensible way to set up multilevel priors. However, if we choose the "independent prior" approach, we miss the opportunity for the intercepts and slopes to inform each other. For example, it is conceivable that subjects who are slow to react already before sleep deprevation are more affected by said depreviation, thus also having a larger slope. Or, conversely, it may be that subjects who are faster at the start are more strongly affected by sleep deprevation.

Admittedly, none of these two directions is very plausible for the sleepstudy data, so let us dive into a little thought experiment where such dependencies are more plausible. Suppose we are analysing how much students learn from a math class by evaluating students math skills both before and after the class. It could be that the math class (and the exams) deal with easy topics, which the good math students already know. Accordingly, the good student's

wouldn't learn much from the class, a pattern that we call a "ceiling effect" in psychology. However, the not-so-good students, who struggled with the exam before the class, have now understood more of the concepts and subsequently improve quite a bit in the exam after class. In this context, smaller intercepts (i.e., lower exam scores before the class) would go hand in hand with higher slopes (i.e., larger differences between the exams after and before the class). Speaking in terms of multilevel models, this would imply a *negative* correlations between varying intercepts and slopes. The same argument can be made for a very difficult math class that only the good students would benefit from, while the math skills of the not-so-good students would remain about the same. This would induce a "floor effect", which implies a *positive* correlation between varying intercepts and slopes. In both of these scenarios, our estimates of the varying coefficients would benefit from a multilevel prior that supports the exchange of information, not only among the coefficients of the same kind (e.g., varying intercepts of different students), but also among the different kinds of varying coefficients (i.e., the varying intercept and slope of the same student).

The way to achieve this is by imposing a *multivariate* normal prior on the pairs (b_{0j}, b_{1j}) of varying intercepts and slopes, with a pair of overall mean coefficients (b_0, b_1) and a covariance matrix Σ_{01} that models the variation of the intercepts and slopes, respectively, as well as their dependency as motivated above:

$$(b_{0j}, b_{1j}) \sim \text{multinormal}((b_0, b_1), \Sigma_{01})$$

Personally, I struggle interpreting variances and I struggle even more interpreting covariances. Fortunately, we can decompose any covariance matrix into a set of standard deviations and a correlation matrix, both being more intuititive quantities for most people I believe. Explaining this for our specific case at hand, the diagnonal elements of Σ_{01} are the variances, which we obtain from the standard deviations σ_0 and σ_1 by simply squaring them. The covariance between the varying intercepts and slopes can be express as the product of the two standard deviations σ_0 and σ_1 and their correlation ρ_{01}. Putting this together, we obtain

$$\Sigma_{01} = \begin{pmatrix} \sigma_0^2 & \sigma_0\sigma_1\rho_{01} \\ \sigma_0\sigma_1\rho_{01} & \sigma_1^2 \end{pmatrix}$$

such that now have σ_0, σ_1, and ρ_{01} as hyperparameters in addition to b_0 and b_1. Comparing this to the "independent prior" approach above, we have actually only added ρ_{01} to the set of hyperparameters. Since we only have two varying coefficients per subject in our case study, the correlation matrix in fact consists of only one relevant element, namely ρ_{01}. If we had more than two coefficients per grouping-level the number of estimated correlation hyperparameters would of course increase correspondingly. We will dive deeper into the specification of priors on correlations and correlation matrices in Section 4.7. For now, it suffices to say that brms applies a uniform prior in $[-1, 1]$ on ρ_{01} by default for this model. The "zero-centered" version used in brms works in basically the same way as before. That is, we set

$$(\tilde{b}_{0j}, \tilde{b}_{1j}) \sim \text{multinormal}((0,0), \Sigma_{01})$$

and then compute the predictor term on the likelihood mean as

$$\mu_n = b_0 + b_1 x_n + \tilde{b}_{0j[n]} + \tilde{b}_{1j[n]} x_n.$$

In brms, we can indicate that we want both varying intercepts and varying slopes of `Days` across subjects by adding `(1 + Days | Subject)` to the model formula, in addition to the overall intercept and overall slope. If we do not want to model the correlation hyperparameters, we can replace the `|` symbol with `||`, that is, `(1 + Days || Subject)` for our present example. Not modeling the correlations will sometimes lead to quite substantial speedups during model estimation so can be a useful shortcut if some models run too slowly. In general, I would recommend to include the correlation parameters though to improve the exchange of information among the varying coefficients. That being said, the complete brms model with correlation (and default priors) looks as follows:

```
fit_sleep3 <- brm(
  Reaction ~ 1 + Days + (1 + Days | Subject),
  data = sleepstudy
)
```

```
summary(fit_sleep3)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Reaction ~ 1 + Days + (1 + Days | Subject)
   Data: sleepstudy (Number of observations: 180)

Multilevel Hyperparameters:
~Subject (Number of levels: 18)
                    Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)          26.86      6.96    15.68    42.45 1.00     1706     2445
sd(Days)                6.49      1.47     4.11     9.88 1.00     1512     2153
cor(Intercept,Days)     0.10      0.30    -0.47     0.70 1.00      908     1667

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept   251.39      7.36   236.73   266.08 1.00     1778     2371
Days         10.35      1.69     6.94    13.73 1.00     1364     1601
```

```
Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    25.89      1.60    23.01    29.33 1.00     3879     2995
```

From the summary output, we see that there is substantial variation in the slopes across subjects, with the posterior mean of σ_1 being around 6.5 and a 95% credible interval spanning roughly between 4 and 10 ms. There appears to be no noteworthy correlation between varying intercepts and slopes as we can see from the results of the correlation hyperparameter ρ_{01}. It's credible interval is quite wide, which makes sense since the correlation is based on only 18 "data points", the 18 subjects included in the dataset. As you may know from your experience with correlations more generally, we usually need much larger datasets to achieve precise correlation estimates, so the high uncertainty here is not surprising from this perspective.

Before we start studying the varying coefficients in detail, let us better understand the implications of adding multilevel structure on the overall intercepts and slopes. In Figure 4.1, we see the conditional effect of `Days` for the complete pooling model vs. the varying intercept-slope model. Two relevant patterns become apparent. First, the mean prediction line is very similar for both models. This is not a coincidence but follows from the fact that all subjects have the same number of observations. We can think of the complete pooling model as obtaining its coefficients by averaging over all 180 observations. The partial pooling model, in contrast, obtains its overall coefficients by averaging over the 18 subjects. If each subject has the same number of obervations, both averages are the same. Yet, the fact that the posterior mean lines are highly similar does not mean that the choice of model becomes irrelevant. That is because, second, the posterior credible intervals of the predictions are very different. Specifically, the partial pooling model implies clearly more uncertainty than the complete pooling model. When I have shown this comparison to students, I have often gotten the question on why higher uncertainty should be considered better and I imagine some of you might ask yourself the same question. The truth is, neither higher nor lower uncertainty are necessarily preferrable. Rather, broadly speaking, it is about finding *just the right amount of uncertainty*, which accurately reflects our knowledge. This can be formalized statistically, where it is commonly known under the term *calibration*. I will provide a practical example of how to study calibration in Section 4.8. I recommend you to also check out Section 3.2 in Bürkner, Scholz, and Radev (2023a), specifically the parts about uncertainty estimation.

We will now turn our attention to the varying coefficients. When we run `coef`, we see that both also the slope posteriors are now varying across subjects:

```
coef(fit_sleep3)$Subject[, , "Days"]
```

```
      Estimate Est.Error       Q2.5     Q97.5
308 19.5272950  2.479121 14.7347208 24.580980
309  1.7370930  2.508660 -3.1702694  6.706188
310  4.8839824  2.472069  0.1526665  9.821735
```

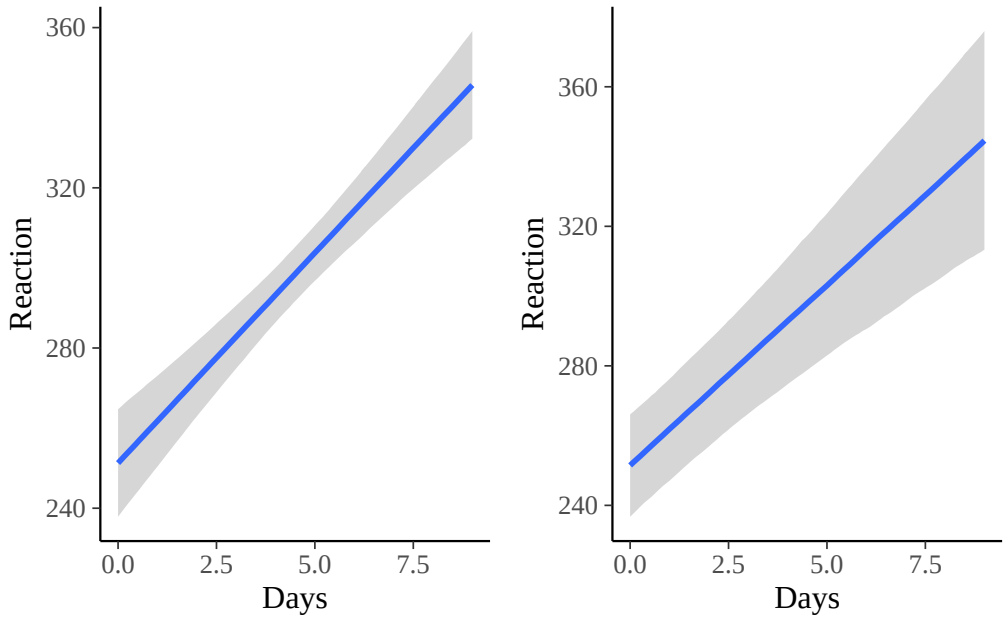
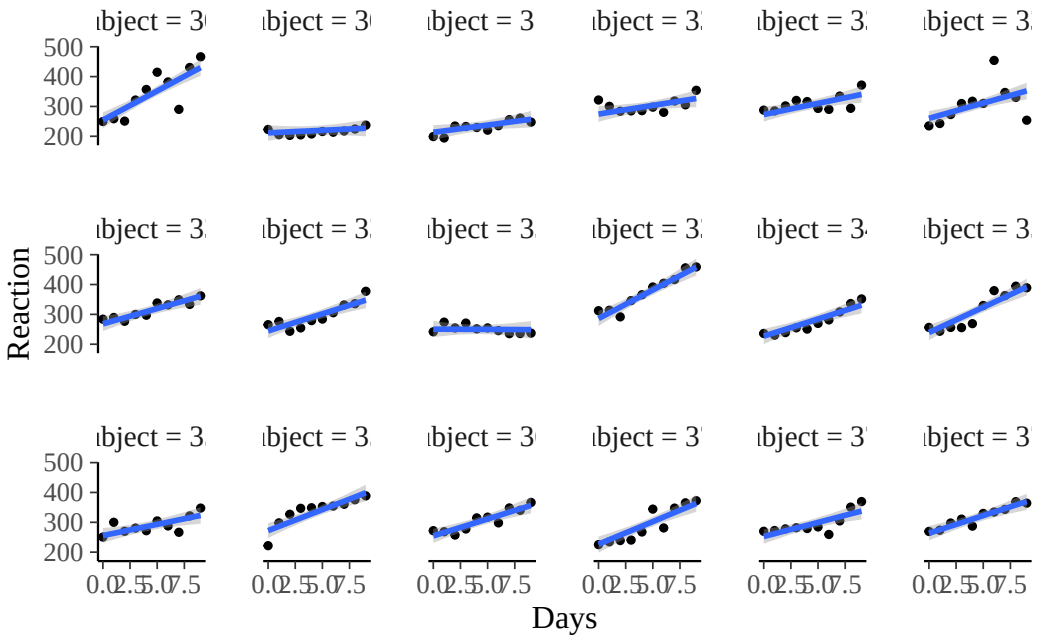


Figure 4.1: Conditional effects of the complete pooling model (left) vs. the partial pooling model (varying intercepts and slopes; right). The blue line indicates the mean predictions while the gray area indicates the 95% credible intervals of the predictions.

```
330  5.7595620  2.554610  0.6996273 10.629359
331  7.4572208  2.425660  2.6777367 12.072865
332 10.1990104  2.352321  5.5103957 14.668787
333 10.2735209  2.394091  5.4959653 14.935536
334 11.4818928  2.354181  6.8432421 16.138818
335 -0.1936086  2.664703 -5.5074099  4.892159
337 19.1188646  2.483403 14.2562326 24.141344
349 11.5503660  2.468425  6.8292754 16.332760
350 16.9516687  2.476767 12.1541822 21.743907
351  7.4676904  2.351740  2.6989864 11.973532
352 14.0408605  2.425689  9.4144842 18.695150
369 11.3395926  2.325526  6.8839859 15.888630
370 15.1563078  2.549139 10.2607253 20.239997
371  9.4768203  2.339056  4.8474928 13.958694
372 11.7310053  2.344721  7.0225056 16.371831
```

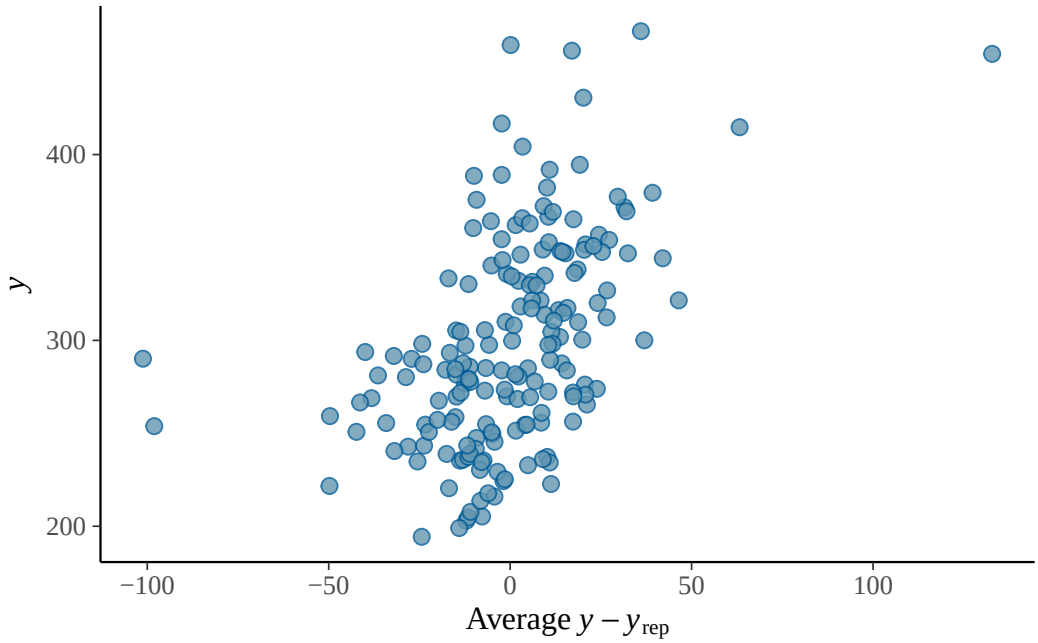
Some subjects, for example Subject 309 or 335, have a posterior mean slope of close to 0ms. That is, their reaction times seem to be basically unaffected by sleep depreviation. Other subjects, for example, Subject 308 or 337, have a slope of about 20ms. That is their reaction times increase by 20ms per day in expectation, which, when extrapolated over the 9 days of sleep depreviation, ultimatiely leads to a quite substantial increase of about 180ms. We can also see this visualized in the per-subject `conditional_effects` plots:

```
ce <- conditional_effects(
  fit_sleep3, conditions = conditions,
  re_formula = NULL
)
plot(ce, ncol = 6, points = TRUE)
```



Clearly, any sensible model of this data should somehow account for this strong variation across subjects. Indeed, the posterior predictive checks are looking much improved compared to the model with only varying intercepts:

```
pp_check(fit_sleep3, "error_scatter_avg")
```



The only remaining issue in model fit we see in this plot are 3 outlying data points, which we can also identify from the above `conditional_effects` plots. It seems that some subjects sometimes had an uncharacteristcally good or bad day, defying the overall trend in their reaction time changes. Based on the available data, there is nothing that provides us with a better understanding why that happens exactly. As a remedy, we could for example change the likelihood from Gaussian to Student-t to make the model less affected by such outliers. I leave this modeling task to you as an exercise.

To compare our models more formally, we again run LOO-CV:

```
loo_sleep3 <- loo(fit_sleep3)
```

```
Warning: Found 3 observations with a pareto_k > 0.7 in model 'fit_sleep3'. We recommend
to set 'moment_match = TRUE' in order to perform moment matching for problematic
observations.
```

We get a warning that our PSIS approximation of LOO-CV has failed for 3 observations, clearly the exact same observations we identified in the above `pp_check` plot. We choose to ignore this warning for now and proceed with interpreting the results. The effective number of parameters evaluates to roughly 34, which is clearly smaller than the nominal 42 parameters we have in our model (36 varying coefficients plus 5 hyperparameters plus the residual standard deviation σ). This is something we usually see in multilevel models as the partial pooling property of

the multilevel priors reduce the de-facto complexity of the model. Let's compare the LOO-CV performance of our three models to each other:

```
loo_compare(loo_sleep1, loo_sleep2, loo_sleep3)
```

```
           elpd_diff se_diff
fit_sleep3   0.0       0.0
fit_sleep2 -24.0      11.7
fit_sleep1 -92.2      21.1
```

The evidence in favor of the varying intercept-slope model `fit_sleep3` being the best is pretty strong, to a degree that it is unlikely caused by PSIS failing for 3 observations. Together with the large differences in varying slopes seen previously, we decide that we have gathered sufficient evidence for choosing the varying intercept-slope model as the best among our 3 candidate models. If the evidence was less clear, we would probably need to run moment matching to improve the trustworthiness of our LOO-CV approximation. Since the results remain basically unchanged, I will just show the code below for your reference.

```
# we need to refit the model while saving all parameters
fit_sleep3 <- update(fit_sleep3, save_pars = save_pars(all = TRUE))
mmloo_sleep3 <- loo_moment_match(fit_sleep3, loo = loo_sleep3)
loo_compare(loo_sleep1, loo_sleep2, mmloo_sleep3)
```

4.6 Predicting coefficients of new levels

One of the really cool things we can do with multilevel models is to predict coefficients of new levels of grouping variables (aka new groups), not previously seen by the model. Let's ask to model to make predictions about days 0 and 9 for Subjects 308 and 309, both present in the original data, and for a new subject that we call `"new"`. It doesn't really matter how we call it. As long as it is not in the list of names in the original data, brms will recognize it as new:

```
newdata <- expand.grid(Days = c(0, 9), Subject = c("308", "309", "new"))
posterior_epred(fit_sleep3, newdata = newdata)
# Error: Levels 'new' of grouping factor 'Subject' cannot be found in the
# fitted model. Consider setting argument 'allow_new_levels' to TRUE.
```

brms throws an error on our first attempt to make predictions for a new group to make sure that we really indented it. Since the predictions of new and old subject will be behave quite differently (see below), I don't want users to accidentally make new level predictions through a typo in the old names. The error message however informs us what to do if we really want

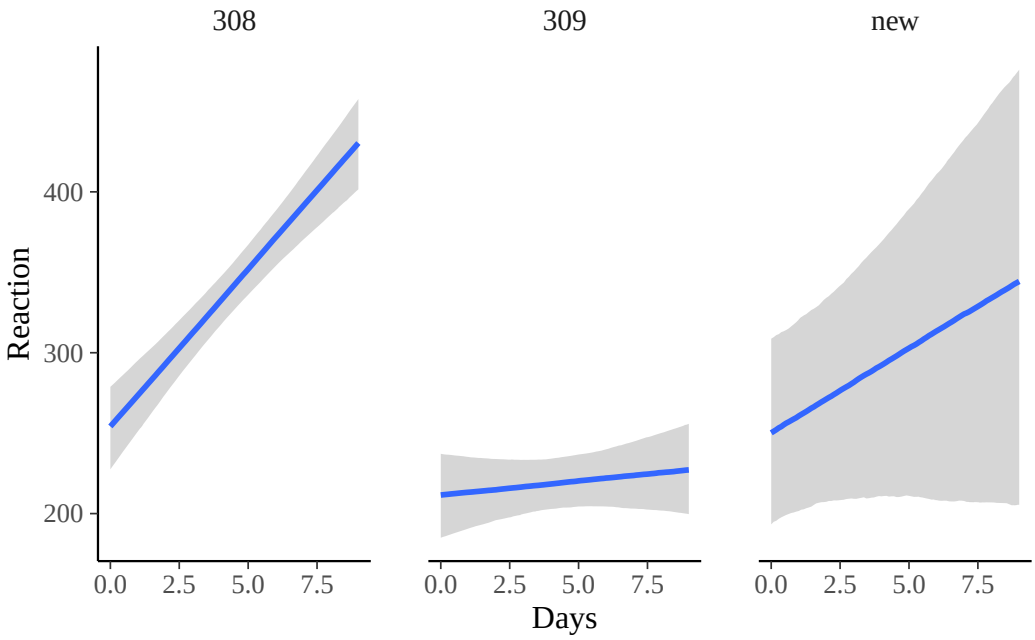
to make predictions for new levels, namely setting `allow_new_levels = TRUE`. Additionally, we can choose among several options of *how* to create varying coefficients for new levels, via argument `sample_new_levels`. We will choose the `sample_new_levels = "gaussian"` option for now. It implies that, for each posterior draw, we will sample varying coefficients from a normal distribution with hyperparameters set to the hyperparameter values in this posterior draw:

```
posterior_epred(
  fit_sleep3, newdata = newdata, allow_new_levels = TRUE,
  sample_new_levels = "gaussian"
) %>%
  posterior_summary() %>%
  cbind(newdata, .)
```

```
  Days Subject Estimate Est.Error     Q2.5    Q97.5
1    0     308 253.9225  12.98362 227.5008 278.8141
2    9     308 430.2240  14.44181 401.5558 457.6983
3    0     309 211.4709  13.36849 185.0033 237.1614
4    9     309 227.3008  14.35047 199.6726 255.8211
5    0     new 251.1462  28.67020 194.2004 308.8900
6    9     new 345.1629  67.95670 212.1779 480.8188
```

The variation in the new-level predictive distribution contains both the uncertainty in the hyperparameters' posterior and the variation implied by sampling from a normal distribution with these hyperparameters. In other words, the new-level predictve distribution contains also variation *across* levels and hence will be much wider than the posterior distributed from levels existing in the original data. This is exactly what we see reflected in the output above. It may help with intuition to see all of this visualized:

```
conds <- data.frame(Subject = c("308", "309", "new"))
rownames(conds) <- conds$Subject
conditional_effects(fit_sleep3, conditions = conds, re_formula = NULL,
                    sample_new_levels = "gaussian")
```



Indeed the new-level predictive distribution covers almost the entire range of the original subjects' predictive distributions. Subjects 308 and 309 are on opposite ends of the distribution of varying coefficients and indeed their predictions are roughly at the borders of the new-level predictive distribution.

The above predictions always combined the posterior uncertainty in the hyperparameter with the subsequent predictive uncertainty on the values of the new varying coefficients. What if we want to only get the latter uncertainty, while keeping hyperparameter values constant? We can solve this in brms by obtaining several predictions for the same posterior draw. Say, we want to obtain 20 new-level predictions for posterior draw 7, we can achieve this as follows:

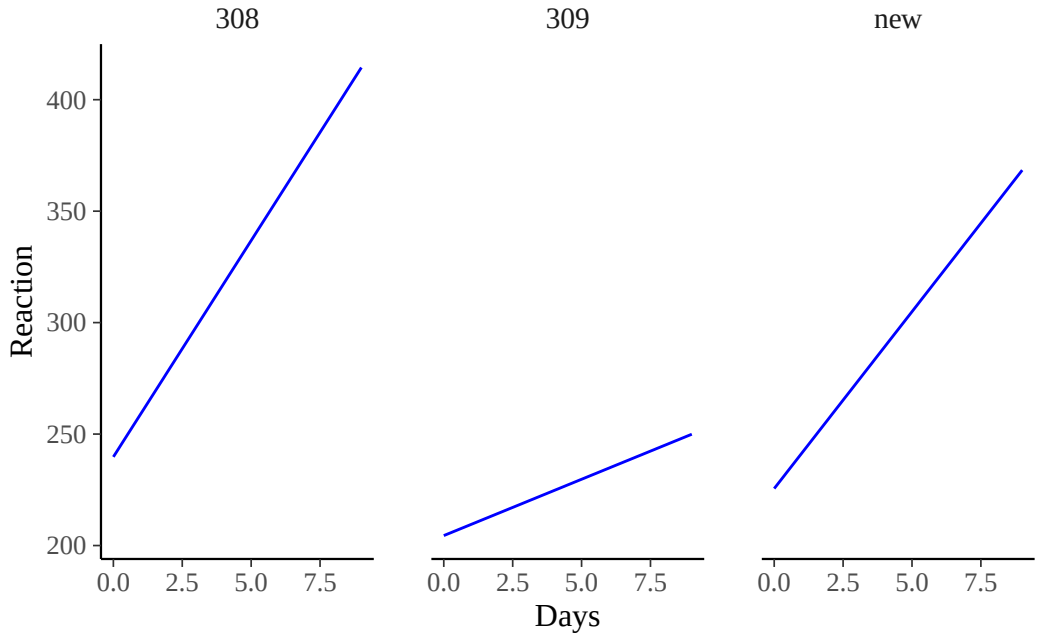
```
posterior_epred(
  fit_sleep3, newdata = newdata, allow_new_levels = TRUE,
  sample_new_levels = "gaussian", draw_ids = rep(7, 20)
) %>%
  posterior_summary() %>%
  cbind(newdata, .)
```

```
  Days Subject Estimate Est.Error     Q2.5    Q97.5
1    0     308 239.7701        NA 239.7701 239.7701
2    9     308 414.4319        NA 414.4319 414.4319
3    0     309 204.4620        NA 204.4620 204.4620
4    9     309 249.9342        NA 249.9342 249.9342
```

```
5    0     new 243.0000         NA 243.0000 243.0000
6    9     new 344.5911         NA 344.5911 344.5911
```

Notice how the old level predictions are constant, since we always used the same posterior draw there. In contrast, the new-level predictions still have variation as a result of sampling from the new-level predictive distribution 20 times, even though hyperparameter values where constant. Again, it helps to see this visualized:

```
set.seed(7332)
ce <- conditional_effects(
  fit_sleep3, conditions = conds, re_formula = NULL,
  sample_new_levels = "gaussian", draw_ids = rep(7, 20),
  spaghetti = TRUE
)
plot(ce, spaghetti_args = list(colour = "blue"),
     line_args = list(colour = alpha("white", 0)))
```

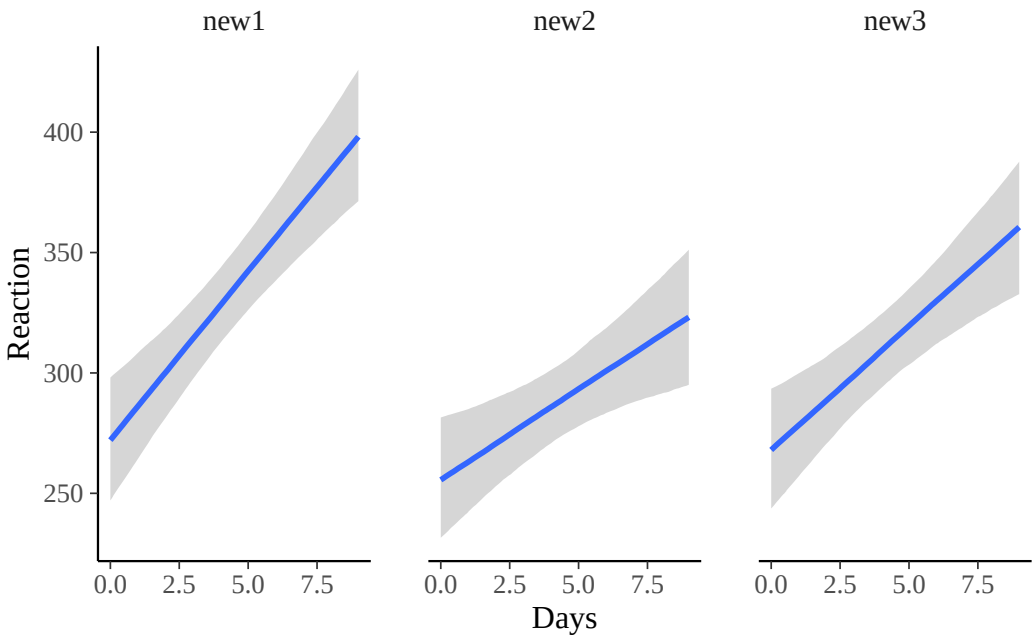


I used the `spaghetti = TRUE` such that we see each prediction line separately rather than having them combined to form an uncertainty band around the mean predictions. Indeed, we see 20 separate lines for the new-level predictions, while there is just a single line for the existing level predictions. The arguments passed to the `plot` method of `conditional_effects` are just some tricks make the individual lines non-transparent, but you will see the same patterns also with the default settings.

Above we have seen the `sample_new_levels = "gaussian"` option throughout, but brms also supports other ways to sample new grouping levels. The default `"uncertainty"` option samples not from a normal distribution but from the posterior draws of the old levels' coefficients, where for each posterior draw, the old level to draw from is newly chosen. The result are predictions quite similar to that of the `"gaussian"` option: If the "across-level" distribution of the old levels' coefficients was indeed normal, the two options would basically be the same. However, it might be that the old levels' coefficients are in fact not normally distributed. After all, the multilevel normal distribution we assumed for them was just a prior after all. In a way, the `"gaussian"` option is a mix of posterior and prior assumptions: We sample the hyperparameters from the posterior but then sample the coefficients from a normal distribution conditional on the posterior hyperparameter values. In contrast, through the `"uncertainty"` option, we sample purely from the posterior, but with the drawback that the "across-level" posterior is only approximated by the coefficients of the levels in the original data. Accordingly, if a grouping factor only contained few levels in the original data, the approximation of "across-level" posterior may be inaccurate.

Lastly, a third `sample_new_levels` option is `"old_levels"`, which means that for every new level, we will randomly choose a *single* old level to take all posterior draws of its coefficients from. Accordingly, the new-level predictions obtained this way will, for each new level, be much more certain than with the `"gaussian"` or `"uncertainty"` options. Let's sample coefficients of 3 different new levels this way and see what happens:

```
set.seed(3552)
conds <- data.frame(Subject = c("new1", "new2", "new3"))
rownames(conds) <- conds$Subject
conditional_effects(fit_sleep3, conditions = conds, re_formula = NULL,
                    sample_new_levels = "old_levels")
```



Indeed, each of the lines is pretty certain while there might be quite some variation across new levels. It may of course also be that we choose the same old level for two new levels, in which case their predictions become the same. Try it out yourself by varying the random seed above until you get two levels to yield the exact same predictions. On my current setup, I see this happening for example when using the seed `3556`.

4.7 Priors on correlation matrices

So far, we have not discussed priors on the correlations between varying coefficients, but this is about to change. As we have mentioned in Section 4.5, a covariance matrix can always be decomposed into a vector of standard deviations and a set of correlations organized in the form of a correlation matrix. As you probably know from your basic stats training, a correlation matrix is symmetric with diagonal elements consisting of 1s and off-diagonal elements (the correlations we care about) being in the interval $[-1, 1]$. For example, in the 3-dimensional case, a correlation matrix Ω looks as follows:

$$\Omega = \begin{pmatrix} 1 & \rho_{12} & \rho_{13} \\ \rho_{21} & 1 & \rho_{23} \\ \rho_{31} & \rho_{32} & 1 \end{pmatrix},$$

where always $\rho_{ij} = \rho_{ji}$. Accordingly, specifying a prior on Ω requires only the lower-diagonal elements to vary since the diagonal elements are 1 and the upper-diagonal elements are the

same as the lower-diagonal ones. Still, defining a prior on correlation matrices is not a trival endavour, since we need to ensure that only *valid* correlation matrices are possible. To put it intuitively, by valid I mean that a correlation matrix could theoretically be obtained from data, or equivalently, that we could use it to simulate data. Mathematically, this translates to the correlation matrix being *positive definite*. I mention this term just that you have heard of it, in case you come across it again at some other point.

Not all correlation matrices are valid even if all included correlations individually are within the interval $[-1, 1]$. Let's make a simple example as an illustration. Think of three variables x_1, x_2, x_3 where x_1 and x_2 are very highly correlated, say, $\rho_{12} = 0.99$. This now restricts which correlations of x_1 and x_3 as well as of x_2 and x_3 are possible. Suppose we also have $\rho_{13} = 0.99$. Then, say, $\rho_{23} = -0.99$ would be impossible (i.e., invalid). Intuitively, this can be understood as follows: If one variable (x_1) is highly positively correlated with two other variables (x_2, x_3), then these two other variables also have to be (relatively) highly positively correlated with each other, certainly not highly negatively correlated. Each of the individual correlations are valid. Only if viewed jointly, they become invalid. As the dimensionality of the correlation matrix grows, the number of such interdependencies between correlations grows, such that the space of valid correlation matrices becomes smaller.

Within the space of valid correlation matrices of given dimension D, we can now choose if we want to favor certain kind of matrices over others. The prior of choice for correlation matrices in brms and Stan is the LKJ prior (Lewandowski, Kurowicka, and Joe 2009), named after the initials of its inventors. This prior has a single positive shape parameter η. If $\eta = 1$ all valid correlation matrices of dimension D are equally likely. That is, $\eta = 1$ implies a "uniform" prior over the space of valid correlation matrices. If $\eta > 1$, correlation matrices with correlations closer to zero are favored and if $\eta < 1$ correlation matrices with correlations further away from zero are favored. In any case, the LKJ prior is always symmetric around zero and has the same marginal distribution for all individual (off-diagonal) correlations that are part of the matrix. For dimensions $K = 2, 3, 9$ and varying η, its marginal distributions for the individual correlations are illustrated in Figure 4.2.

For $D = 2$, the uniform case $\eta = 1$ indeed means that all correlations in $[-1, 1]$ are equally likely. This makes sense since, for $D = 2$, the correlation matrix only contains a single correlation that is free to vary. Still for $D = 2$, if we set $\eta < 1$ or $\eta > 1$, we see the above described patterns of a U-shaped or an inverse U-shaped prior, respectively. This changes as soon as $D > 2$. Already at $D = 3$, we see how the supposably "uniform" choice of $\eta = 1$ implies very much non-uniform marginal prior where highly positive or negative correlations become less likely. As the simple example above illustrates, this makes total sense, but it is certainly not the most intuitive behavior. Incidentally, $\eta = 0.5$ now implies a uniform marginal prior for $D = 3$. At $D = 9$, we see an even stronger tendency towards zero correlations, where there is even barely any difference anymore between the hyperparameter values of $\eta = 0.5, 1, 2$.

As a user, you will rarely have to worry about the specifics of the LKJ prior or valid correlation matrices more generally. In my experience, setting $\eta = 1$ is usually a safe choice and also the default in brms. If we want to make more extreme correlations a priori less likely, we may

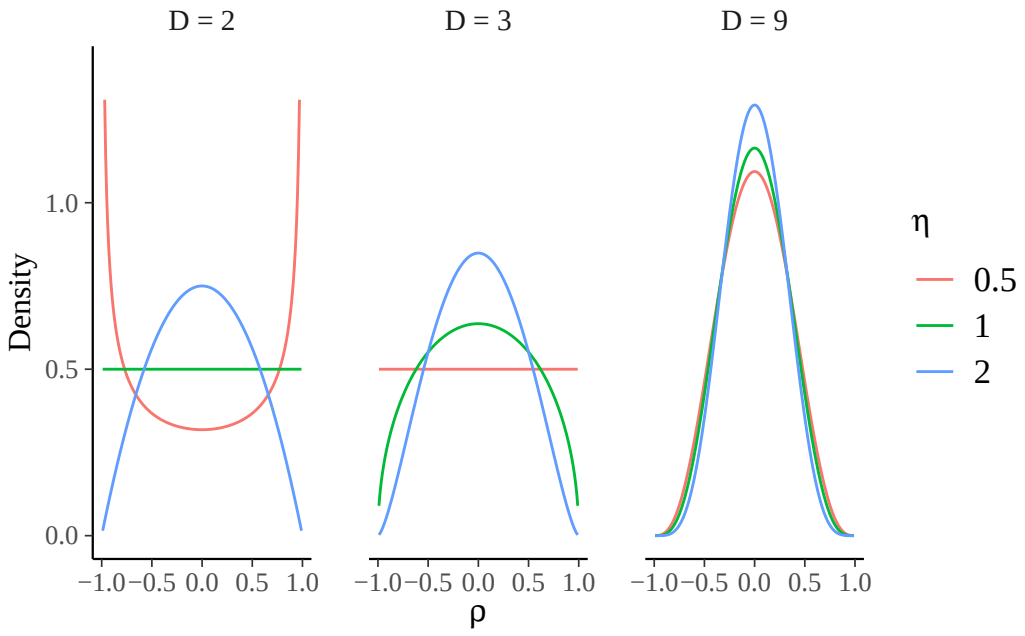


Figure 4.2

choose something like $\eta = 2$ or even $\eta = 4$ but this will only have a noticable effect on the marginal prior distributions in lower dimensions as the $D = 9$ case in Figure 4.2 has shown us.

In brms, we can set LKJ priors on correlation matrices via the pattern `prior(lkj(<eta>), class = "cor")`. The dimensions of the correlation matrices are automatically inferred from the model and do not need to specified in the prior argument. As an exercise, refit the varying intercept-slope model `fit_sleep3` with varying LKJ priors (say, $\eta = 0.5, 1, 2, 4$) and investigate how the posterior on the correlation between the varying intercepts and slopes changes.

4.8 Simulation study of Type 1 errors

I hope I have convinced you so far that multilevel models are a reasonable choice for the `sleepstudy` data, at least more reasonable than ignoring the repeated measurements per subject altogether. But what implications does it have if we decide to ignore the multilevel structure still? One thing I know a lot of researchers are caring about are Type 1 error rates in the context of null hypothesis significance testing. That is, given a truely null effect, how likely is it that the statistical model or test will incorrectly lead us to conclude that there is a non-null effect?

For our example this concretly means the following: Suppose there truely was no effect of sleep deprevation on the reaction times (i.e., $b_1^* = 0$), how likely it is that the model would lead us to conclude that there is an effect of sleep depreviation? This is a question at the heart of frequentist statistics, but I see no issues using Bayesian models to answer frequentist questions. There is no quarantee that a Bayesian model has correct frequentist calibration but neither is there a guarantee that a frequentist model has correct frequentist calibration, except in very simple scenarios that are unlikely in reality.

Of course, we should still understand the frequentist properties of our Bayesian models before we use them for this purpose. To make this more concrete, if our models have a correct frequantist calibration for the parameter b_1 we should, for example, see that its 95% posterior credible interval contains the true value of $b_1^* = 0$ in 95% of the datasets generated from this true value. We cannot use real datasets for the purpose of checking the calibration of any (latent) model parameter, since we will never know its true value exactly; even if we could argue that the parameter itself would somehow "exist" for this dataset. Accordingly, we need simulated data. And not just one simulated dataset, but many of them, since frequentist calibration makes a statement about the data-generating process, not about any specific dataset. In other words, we will need a simulation study.

First, we set up a function for our assumed data-generating process. Here, we will assume that the varying intercept-slope model is indeed the "true" model. To make the code a bit easier to read, we will only allow for equal number of observations per subject and assume the correlation between varying intercepts and slopes to be always zero. Since this roughly resembles the situation in our real data, I will be just fine for our purposes.

```
# simulate datasets that resemble the structure of the sleepstudy data
sim_sleepstudy_data <- function(beta0, beta1, tau0, tau1, sigma,
                                ngroups = 18, nobs_per_group = 10) {
  u0 <- rnorm(ngroups, 0, tau0)
  u0 <- rep(u0, each = nobs_per_group)
  u1 <- rnorm(ngroups, 0, tau1)
  u1 <- rep(u1, each = nobs_per_group)
  Subject <- factor(rep(1:ngroups, each = nobs_per_group))
  Days <- rep(0:(nobs_per_group - 1), each = ngroups)
  e <- rnorm(ngroups * nobs_per_group, 0, sigma)
  Reaction <- beta0 + u0 + (beta1 + u1) * Days + e
  data.frame(Reaction, Days, Subject)
}
```

Second, we set up a function that fits the models of interest to a simulated dataset. We chose to focus on comparing the complete pooling model `fit_sleep1` and the varying intercept-slope models `fit_sleep3`:

```
# fit the models of interest to a simulated dataset
# the simulation index j is unused here but still provided
# as an argument to prevent it from being passed further
# returns a matrix of posterior draws of b1 with
# one column per fitted model and one row per posterior draw
sim_sleepstudy_models <- function(j, ...) {
  dat <- sim_sleepstudy_data(...)

  fit1 <- update(fit_sleep1, newdata = dat, chains = 1)
  draws1 <- as.data.frame(fit1, variable = "b_Days")$b_Days

  fit2 <- update(fit_sleep3, newdata = dat, chains = 1)
  draws2 <- as.data.frame(fit2, variable = "b_Days")$b_Days

  return(cbind(draws1, draws2))
}
```

Third, we will actually run our simulations. Since, for each simulated dataset, we are fitting multiple Bayesian models, this may take a while. It is thus highly beneficial to parallelize the code in order to run multiple simulation trials at the same time. There are many ways to set up parallel code in R but I personally prefer using the `future` package as it abstracts away a lot of the complications that arise in parallel computing.

```
library(future)
library(future.apply)
```

We will run the simulations for 250 trials, that is, 250 simulated datasets using 8 cpu cores (called `workers` here). When running these simulations on your own machine, make sure that you do not request more cpu cores than you have on your machine; ideally one or two cores less if you plan on doing anything else on your machine while the simulations run.

```
nsim <- 250
# set up a parallel environment to run futures in
plan(multisession, workers = 8)
# future_lapply is like lapply just with parallel execution
sim_results <- future_lapply(
  1:nsim, sim_sleepstudy_models,
  beta0 = 250, beta1 = 0, tau0 = 25,
  tau1 = 7, sigma = 25,
  future.seed = TRUE
)
```

```
# go back to sequential evaluation after finishing the simulations
plan(sequential)
```

Lastly we will transform and summarize the results. First, we will look at the histogram of the posterior quantile in which the true parameter value $b_0^* = 0$ lies. That is, for each simulation trial, we compute how many (in percent) of the posterior draws are smaller than the true value:

```
# combine the results into an array
sim_results_array <- abind::abind(sim_results, along = 3)
# compute posterior quantiles of the true parameter value
true_beta1 <- 0
true_beta1_quantiles_1 <- colMeans(sim_results_array[, 1, ] < true_beta1)
true_beta1_quantiles_2 <- colMeans(sim_results_array[, 2, ] < true_beta1)
```

Under perfect frequentist calibration, the distribution of these quantiles across simulation trials should be uniform. Any deviation from uniformity beyond mere chance provides evidence for miscalibration of the posterior. So let's check out the histograms:

```
histogram(true_beta1_quantiles_1, bins = 10) +
  histogram(true_beta1_quantiles_2, bins = 10)
```

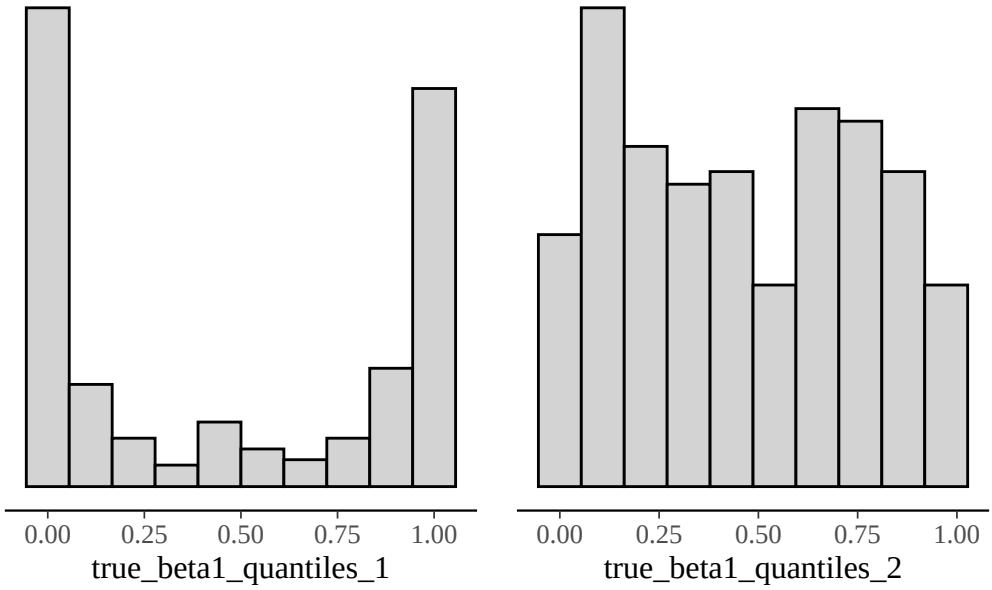


Figure 4.3: Histograms of the true-value posterior quantiles of slope parameter b_1 for the complete pooling model (left) vs. the partial pooling model (varying intercepts and slopes; right). Under good calibration these histogram are expected to be approximately uniform.

On the left-hand side, for the complete pooling model, we see a strong deviation from uniformity in the way that very small quantiles and very high quantiles are more likely than would be expected under good calibration. This means that the posterior of b_1 under this model is too narrow such that the true value of $b_1^* = 0$ is often in the tails of the posterior. For the varying intercept-slope model, results look more uniform but it is a bit unclear if the variation we see indicate some systematic miscalibration or just random noise due to our limited simulation budget of 250 simulation trials and binning artifacts. We could add some confidence intervals for the histogram bars, but we will opt for a different method right away. Instead of histograms, we will plot the empirical cumulative distribution function (ECDF) of the above computed quantiles. Under perfect frequentist calibration, the ECDF should resemble the CDF of a uniform distributions, that is, simply a diagonal line. If the deviation from uniformity is so large that is cannot be explained by chance, this points to some kind of miscalibration. For ECDFs we can obtain simultanous confidence envelopes that take into account the ECDF curve as a whole (Säilynoja, Bürkner, and Vehtari 2022) and we have access to this kind of plot via the `bayesplot` package:

```
true_betas1 <- rep(0, nsim)
ppc_pit_ecdf(true_betas1, sim_results_array[, 1, ]) +
  ppc_pit_ecdf(true_betas1, sim_results_array[, 2, ])
```

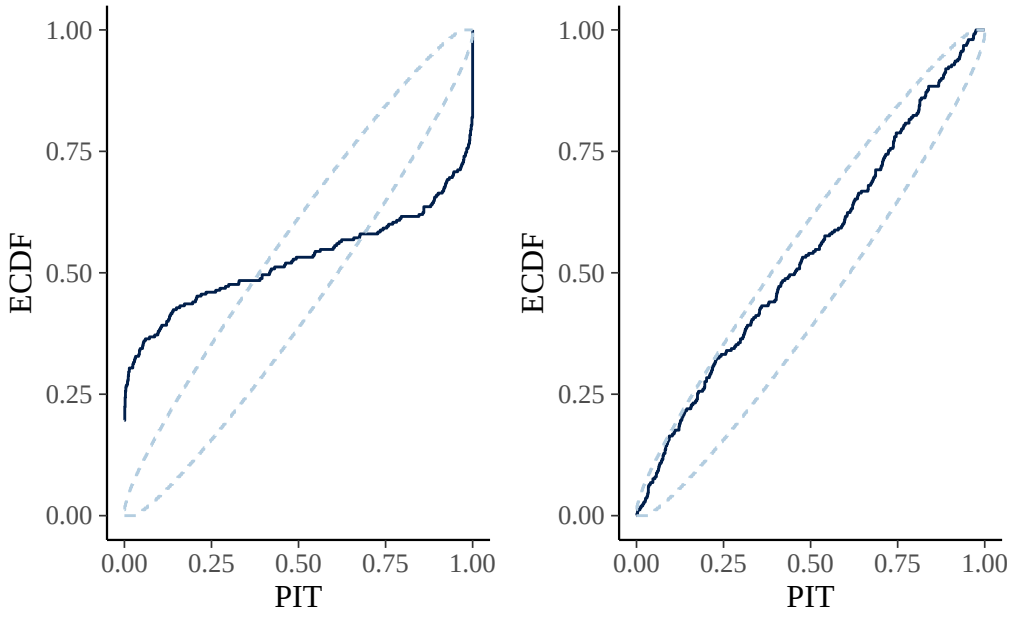


Figure 4.4: ECDF plots of the true-value posterior quantiles of slope parameter b_1 for the complete pooling model (left) vs. the partial pooling model (varying intercepts and slopes; right). Under good calibration the ECDF line (dark blue) is expected to lie within the confidence envelope (light blue lines).

In the above ECDF plots, the dark blue lines indicates the ECDF and the line blue lines indicate the borders of the region within which the ECDF will highly likely fall (with probability 99% by default), if the underlying distribution was indeed uniform. Confirming what we have seen in the histograms, the complete pooling model is clearly miscalibrated for small and large parameter values, while the varying intercept-slope model shows overall good calibration.

These results are very closely related to the expected Type 1 error rates. Namely, if we see too many extreme true-value quantiles than would be expected under uniformity, we will have an inflated Type 1 error rate. To make this more concrete, suppose we would use central 95% credible intervals to indicate "significance": If the value under the null hypothesis (here $b_1^* = 0$) was outside of the credible interval, we would declare significance. To check how often this was the case in our simulations, all we have to do is to check how many true-value quantiles are smaller than 2.5% or larger than 97.5%. For the first model, we compute:

```r
mean(true_beta1_quantiles_1 < 0.025 | true_beta1_quantiles_1 > 0.975)
```

```
[1] 0.572
```

That is, we have a Type 1 error rate of about 50%(!), while it should nominally be around only 5%. Such kind of extreme error rate inflations are not uncommon when ignoring multilevel structure and should serve as a clear warning for anyone ignoring or not sufficiently thinking about such structure in their data. In contrast, the same computation for the varying intercept-slope model yields an error rate of 0.03, which is very close to the nominal 5%. This is merely a sanity check since we build our simulations assuming said model to be true. If there was relavent signal in the data not captured by this model, it would also be miscalibrated of course. The point I want to make here was merely to showcase what kind of bad things can happen if you ignore relevant structure in your data.

4.9 No pooling

Clearly, we should not ignore multilevel structure in our data. But what happens if we just model a separate intercept and separate slope for each subject without a dependency inducing prior, almost as if we modeled each subject's data completely separately? This is what we refer to as *no pooling* model. Such a model will just require basic R formula syntax without any multilevel terms and we can specify it as follows:

```r
fit_sleep4 <- brm(
  Reaction ~ 0 + Subject + Subject:Days,
  data = sleepstudy
)
```

```r
summary(fit_sleep4)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Reaction ~ 0 + Subject + Subject:Days
   Data: sleepstudy (Number of observations: 180)

Regression Coefficients:
               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Subject308        244.07     15.40   213.82   274.73 1.00     7494     3008
Subject309        205.10     15.35   176.04   234.57 1.00     7626     3002
Subject310        203.46     15.18   174.06   232.63 1.00     7845     3506
Subject330        289.92     15.52   259.46   319.65 1.00     7965     3052
```

```
Subject331        285.69     15.46   255.77   315.72 1.00     6711     3213
Subject332        264.18     14.76   235.57   292.94 1.00     7514     3227
Subject333        275.16     14.86   246.92   304.10 1.00     7247     3267
Subject334        240.22     15.34   210.98   270.26 1.00     8010     2822
Subject335        263.24     14.49   234.46   291.31 1.00     6987     3224
Subject337        290.21     15.19   261.48   319.74 1.00     7301     3104
Subject349        215.40     14.93   185.40   244.99 1.00     7330     2944
Subject350        225.85     15.52   195.06   255.53 1.00     8127     3067
Subject351        261.09     15.46   230.27   291.58 1.00     5700     2419
Subject352        276.07     15.26   246.12   305.94 1.00     8564     3254
Subject369        255.23     15.59   225.13   285.35 1.00     7133     3205
Subject370        210.43     15.43   179.82   240.02 1.00     7363     2996
Subject371        253.70     14.80   224.21   282.85 1.00     6656     3061
Subject372        266.80     15.03   237.00   295.98 1.00     6802     2553
Subject308:Days    21.78      2.91    16.02    27.62 1.00     7860     2830
Subject309:Days     2.26      2.87    -3.39     7.71 1.00     7869     2931
Subject310:Days     6.13      2.84     0.56    11.79 1.00     8146     3686
Subject330:Days     2.98      2.94    -2.70     8.62 1.00     7065     2915
Subject331:Days     5.25      2.90    -0.46    10.92 1.00     6680     3104
Subject332:Days     9.58      2.81     4.23    15.09 1.00     8022     3094
Subject333:Days     9.10      2.75     3.62    14.46 1.00     7005     3330
Subject334:Days    12.25      2.88     6.68    17.75 1.00     8948     2888
Subject335:Days    -2.93      2.76    -8.27     2.39 1.00     7834     3249
Subject337:Days    18.98      2.86    13.33    24.52 1.00     7048     2664
Subject349:Days    13.44      2.82     7.99    19.10 1.00     7817     3102
Subject350:Days    19.50      2.95    13.69    25.26 1.00     7717     3278
Subject351:Days     6.46      2.90     0.71    12.20 1.00     6229     2805
Subject352:Days    13.63      2.86     8.04    19.36 1.00     7749     2979
Subject369:Days    11.26      2.92     5.44    16.94 1.00     7407     3017
Subject370:Days    18.03      2.90    12.41    23.83 1.00     6823     2975
Subject371:Days     9.21      2.78     3.75    14.69 1.00     6524     3018
Subject372:Days    11.33      2.79     5.88    16.94 1.00     7085     3113

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    25.80      1.55    23.05    29.11 1.00     3710     3155
```

The only thing that distinquishes this model from just a separate model per subject is that we estimate a single `sigma` parameter across subjects. Remember when we found small positive correlations between varying coefficients that I told you were due to partial pooling? Let's check out the posterior correlations again with the no pooling model.

```
vars_intercepts <- paste0("b_Subject", unique(sleepstudy$Subject))
coef_sleep4 <- as.matrix(fit_sleep4, variable = vars_intercepts)
print(cor(coef_sleep4[, 1:5]), digits = 2)
```

```
             b_Subject308 b_Subject309 b_Subject310 b_Subject330 b_Subject331
```

```
b_Subject308       1.0000      -0.0048      -0.0200      -0.0271       -0.011
b_Subject309      -0.0048       1.0000      -0.0370       0.0192       -0.022
b_Subject310      -0.0200      -0.0370       1.0000       0.0063       -0.042
b_Subject330      -0.0271       0.0192       0.0063       1.0000       -0.039
b_Subject331      -0.0107      -0.0223      -0.0417      -0.0394        1.000
```

Indeed, virtually zero correlations, with the minimal deviations fully attributible to MCMC error. But there is another, arguably much more important effect of partial pooling on the coefficient estimates, namely that these priors induce *shrinkage* towards the overall mean. In Figure 4.5, we can see how shrinkage plays out for our models. Indeed we see noticably differences between the no pooling and the partial pooling estimates, with differences that tend to be higher for subjects whose coefficients are further away from the overall mean. For some subject this shrinkage is actually quite substantial. For other subjects, close the the overall mean, there is barely any shrinkage at all. If we look at the plot in more detail, we see some curious details. For example, the two subjects' estimates on the lower left corner (small intercepts, small slopes) seem to be moving *further away* from the overall mean, at least in the Y-direction. That is, through partial pooling their slopes are estimated to be even smaller than in the no pooling model. This is admittedly unintuitive, but does not indicate a bug in our multilevel model, rather a misconception in the intuition. Partial pooling will induce *joint* shrinkage, that is shrinkage in the overall coefficient vector, which may mean that for some individual coefficients there is no shrinkage at all, or even a "reversed" shrinkage as is the case for the slopes of the two subjects in question.

What influence do the multilevel priors have on prediction? Usually, they will improve out-of-sample predictive performance compared to the respective no pooling models. The differences in predictive performance are larger if groups have few observations relative to the number of coefficients estimated per group. The lower the number of observations per group or the higher their number of coefficients, the more likely the no pooling model will overfit (i.e., treat noise as signal), thus reducing its out-of-sample predictive performance. In our case, we have 10 observations and only 2 coefficient per subject. In this case, the improvements in predictions induced via partial pooling are likely not that large. Let's check it out:

```
loo_sleep4 <- loo(fit_sleep4)
mmloo_sleep4 <- loo_moment_match(fit_sleep4, loo = loo_sleep4)
loo_compare(mmloo_sleep3, mmloo_sleep4)
```

```
           elpd_diff se_diff
fit_sleep3  0.0       0.0
fit_sleep4 -3.4       2.8
```

Indeed, while the partial pooling model provides a little better predictions in terms of ELPD difference, the corresponding standard error is of about the same size. Accordingly, the evidence for better out-of-sample predictions of the partial pooling vs. the no pooling model is

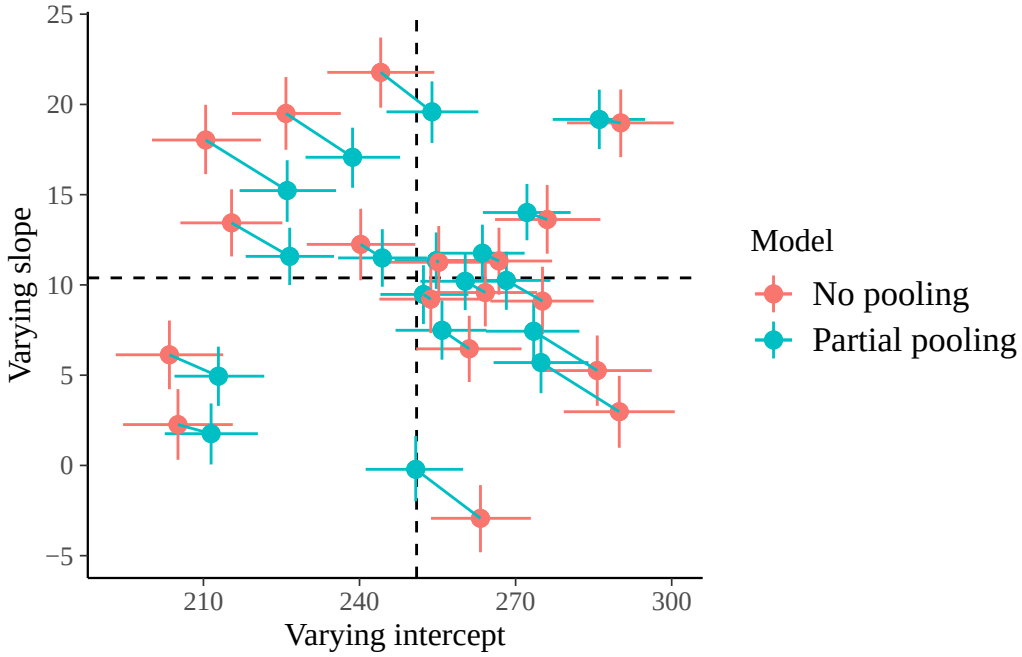


Figure 4.5: Varying coefficients of the no pooling model (red) vs. the partial pooling model (varying intercepts and slopes; blue). Points indicate posterior means and lines indicate central 50% credible intervals.

small in our case. This would drastically change if we decreased the number of observations per subject. For example, when we fit the two models to a reduced dataset containing only data from Days 0, 3, 6, and 9 (i.e., four observations per subject), the LOO-CV performance of the partial pooling model becomes way better than that of the no pooling model. Try it out yourself as an exercise.

4.10 Models with more than two levels

So far, we have seen only how to handle a single grouping factor, but brms allows modeling, in principle, arbitrarily many of them. To illustrate this, let's pretend that the sleep study was conducated in three different sleep labs, each of which monitored 6 of the 18 subjects:

```
sleepstudy <- sleepstudy %>%
  mutate(Lab = rep(c("A", "B", "C"), each = 6 * 10))
```

We may not know exactly how the sleep labs differ from each other and how that might affect the subjects' performance during the study. One example for a potential mechanims driving such variation could be that the computer systems on which subjects perform the reaction time tasks differ across labs. If such a difference is a priori plausible, then, without accounting for the labs in the model, the assumption of a priori exchangable of subjects is no longer justified. We may not know which lab leads to faster or slower reaction times. Yet, we cannot arbitrarly change the order of the subjects anymore without changing our prior assumptions, since changing the order may imply subjects "switching labs".

In an effort to restore a priori exchangable, we decide to model the intercept as also varying across labs. Building on and extending the varying intercept-slope model from Section 4.5, the observation level likelihood mean μ_n is then computed as

$$\mu_n = b_0 + b_1 x_n + \tilde{b}_{0j[n]} + \tilde{b}_{1j[n]} x_n + \tilde{\alpha}_{0l[n]}$$

where the lab-specific intercepts $\tilde{\alpha}_{0l}$ have the usual zero-centered normal prior

$$\tilde{\alpha}_{0l} \sim \text{normal}(0, \gamma_0),$$

with standard deviation parameter γ_0 also estimated from the data. In brms, we achieve this by simply adding `(1 | Lab)` to the model formula.

In many intro books and papers about hierarchical/multilevel models, you may see the more-than-two-level case being introduced a bit differently, by defining equations not only at observation level as done above, but also at all the other levels. For our example, this would mean not writing the varying coefficients of labs in the same equation as the coefficients of subjects,

but rather writing the coefficients of subjects *as a function of* the coefficients of labs (since subjects are nested in levels). I have always struggled with this "multi-equation" formulation of multilevel models since the notation needed to adequately express it quickly blows up. Understanding that, due to linearity, all of those equations can be just written equivalenty as a single equation, has made it much simpler for me to understand and reason about multilevel models. The "single-equation" formulation also generalizes much better to even more complicated setting as we will see in later chapters. It is used consistently in brms.

Back to our sleepstudy model: Since `Lab` has only three levels, its standard deviation parameter γ_0 will be hard to estimate from data alone (think of estimating a standard deviation from just three data points). Accordingly, we should help the model a bit by setting an informative prior on γ_0. Let's say, we assume the baseline reaction time differences between labs to be small, say, unlikely to be larger than 60 ms between the labs with the fastest and the lab with the slowest reaction times. If we interpret these 60 ms as the 95% uncertainty interval of the varying intercepts prior normal$(0, \gamma_0)$ then this implies γ_0 should not be larger than 15 ms, since roughly 95% of the values of a normal distribution lie in the interval [Mean - 2 SD, Mean + 2 SD]. Of course, γ_0 might very well be smaller than 15 ms. We can encode such assumptions with various kind of priors but one straightforward choice is to use a half-normal prior with standard deviation equal to half the maximal expected γ_0:

$$\gamma_0 \sim \text{normal}_+(0, 7.5)$$

Putting it all together, results in the following brms model:

```
fit_sleep5 <- brm(
  Reaction ~ 1 + Days + (1 + Days | Subject) + (1 | Lab),
  data = sleepstudy,
  prior = prior(normal(0, 7.5), class = "sd", group = "Lab"),
  sample_prior = "yes"
)
```

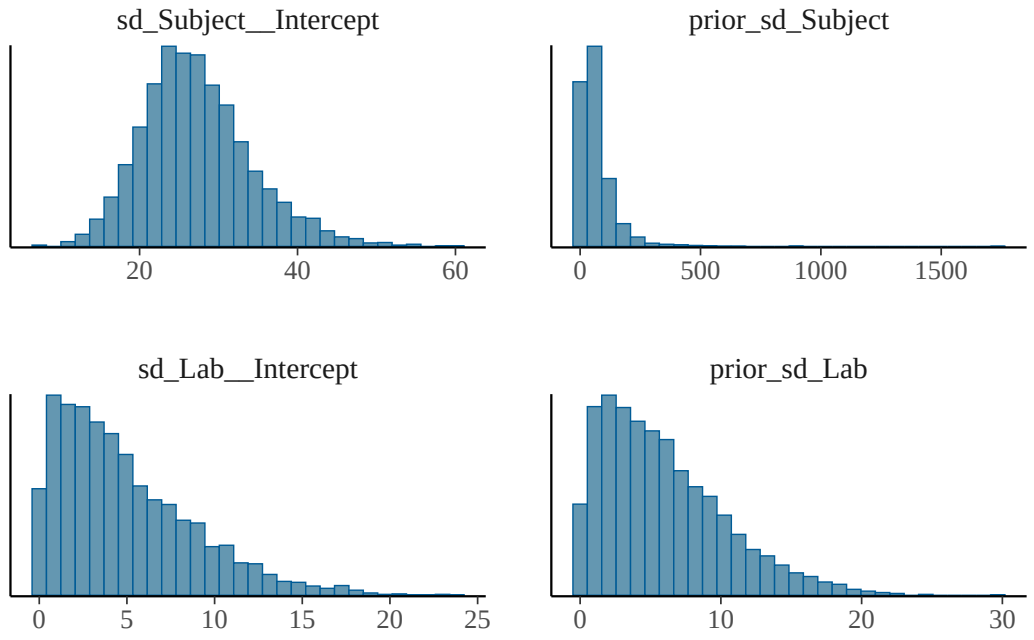
```
summary(fit_sleep5)
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Reaction ~ 1 + Days + (1 + Days | Subject) + (1 | Lab)
   Data: sleepstudy (Number of observations: 180)

Multilevel Hyperparameters:
~Lab (Number of levels: 3)
              Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)     4.95      3.91     0.18    14.65 1.00     2717     1980
```

```
~Subject (Number of levels: 18)
                  Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sd(Intercept)        27.27      6.95    15.45    42.95 1.00     1755     2089
sd(Days)              6.54      1.52     4.22    10.09 1.00     1604     1967
cor(Intercept,Days)   0.08      0.30    -0.47     0.67 1.00     1028     1644

Regression Coefficients:
          Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept   251.16      8.22   235.16   267.76 1.00     1868     2139
Days         10.38      1.72     6.92    13.76 1.00     1324     1836

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    25.92      1.57    23.09    29.15 1.00     3676     3126
```

We added the `sample_prior = "yes"` option to also obtain prior draws in addition to the posterior draws, which enables easy comparison of prior and posterior:

```
vars_subject <- c("sd_Subject__Intercept", "prior_sd_Subject")
vars_lab <- c("sd_Lab__Intercept", "prior_sd_Lab")
mcmc_plot(fit_sleep5, type = "hist", variable = c(vars_subject, vars_lab))
```



For the SD parameter σ_0 of the subject-specific intercepts, we have implicitely used the default prior of brms, which is a half-Student-t prior with small degrees of freedom and data-adaptive scale. Comparing the prior and posterior of σ_0, we clearly see that we have learned a lot about σ_0. In comparison, there is very little difference in the prior and posterior of the SD parameter γ_0 of the lab-specific intercepts. For once, we already started with a much more informative prior and second, we only have the intercepts of three labs (i.e., three "data points") to inform the posterior. Had we used a more weakly informative prior, we would have seen stronger prior-posterior differences for γ_0, yet not nearly as much as for σ_0. I suggest you to try out different priors on γ_0 and σ_0 as an exercise.

As we see from the model formula above, it is syntactically not difficult to extend multilevel models to as many levels as you like. The decisions we have to make primarily involve (a) to identify which relevant grouping factors are present in the data and (b) to decide which of the predictors' coefficients can and should be modeled as varying across which grouping factors. For the present model, we have made the choice to let the intercept vary across both subjects and labs while we allowed the slopes of the `Days` predictor to only vary across subjects. This is choice that may or may not be justified. We could have as well chosen to model the slopes of `Days` to vary across both subjects and labs or even only across labs. This is not to say that all of these choices are sensible, of course. In fact, the there has been much debate about the process that leads us to decide which multilevel terms to include or to exclude, a topic I will come back to in **?@sec-glmms**.

4.11 Different covariance matrices by group

Let us expand our model in yet another direction by pretending that each subject belongs to either one of two groups: Those who like to sleep long and those who like to sleep short. Here is how I chose to assign subjects to these hypothetical two groups:

```
sleepstudy <- sleepstudy %>%
  mutate(
    Sleeper = ifelse(
      Subject %in% c(308, 331, 332, 337, 349, 350, 352, 370, 372),
      "long", "short"
    )
  )
```

It is likely that people who tend to sleep longer compared to shorter will have more trouble with sleep deprivation. We can express this statistically by modeling an interaction effect of `Days` and `Sleeper`, in addition to their individual main effects. The main effects and interaction can be conveniently written as `Days * Sleeper` in the model formula.

With the overall coefficients settled, we turn to the varying coefficients. The first question that may come to mind is whether it makes sense to model subject-specified coefficients of `Sleeper` as well? This would come down to the question of asking whether the habit of sleeping long vs. short affects different people differently. It is a potentially interesting question but cannot be answered by the present data since each subject is classfied as either a long or a short sleeper. In other words the predictor variable is constant within each of the grouping levels. Accordingly, it is impossible to statistically infer any variation of the sleeper effects across subjects. If we want to put this into a single sentence to remember the principle behind it, it goes as follows: "If you want to model a predictor's coefficient as varying *between* the levels of a grouping factor, the predictor's values need to vary *within* said levels."

So modeling varying coefficients for `Sleeper` is not possible for the given data. But this doesn't mean `Sleeper` is necessarily irrelevant for the multilevel terms of our model. For example, it is conceivable that short sleepers are more similar to each other than long sleepers when it comes to how they handle sleep deprivation. In other words, we would expect a smaller SD of the varying slopes in the short vs. the long sleeper group. If we have reasons to believe something like this to be the case, we again are in a situation where a priori exchangability of all subjects is no longer justified, as changing the subjects' ordering may change their sleeper status in a way not fully accounted for by the overall coefficients. The formula syntax in brms allows to handle such a case, using the `gr` function on the right-hand side of multilevel terms. By replacing `Subject` in the `(Days | Subject)` term with `gr(Subject, by = Sleeper)` we tell the model that we want the multilevel hyperparameters (standard deviations and correlations) to vary by the factors of the `by` variable, here by the `Sleeper` categories:

```
fit_sleep6 <- brm(
  Reaction ~ Days * Sleeper + (Days | gr(Subject, by = Sleeper)),
  data = sleepstudy
)
```

```
summary(fit_sleep6)
```

```
Warning: There were 3 divergent transitions after warmup. Increasing adapt_delta above
0.8 may help. See
http://mc-stan.org/misc/warnings.html#divergent-transitions-after-warmup
```

```
 Family: gaussian
  Links: mu = identity; sigma = identity
Formula: Reaction ~ Days * Sleeper + (Days | gr(Subject, by = Sleeper))
   Data: sleepstudy (Number of observations: 180)

Multilevel Hyperparameters:
~Subject (Number of levels: 18)
```

```
                                                Estimate Est.Error l-95% CI u-95% CI Rhat
sd(Intercept:Sleeperlong)                          31.13     11.82    13.28    60.17 1.00
sd(Days:Sleeperlong)                                5.61      2.35     2.11    11.05 1.00
sd(Intercept:Sleepershort)                         30.33     11.37    13.08    58.18 1.00
sd(Days:Sleepershort)                               5.03      2.23     1.72    10.54 1.00
cor(Intercept:Sleeperlong,Days:Sleeperlong)        -0.12      0.40    -0.77     0.74 1.00
cor(Intercept:Sleepershort,Days:Sleepershort)       0.27      0.42    -0.56     0.95 1.00
                                                Bulk_ESS Tail_ESS
sd(Intercept:Sleeperlong)                           1524     1644
sd(Days:Sleeperlong)                                1448     1936
sd(Intercept:Sleepershort)                          1842     2411
sd(Days:Sleepershort)                               1562     1960
cor(Intercept:Sleeperlong,Days:Sleeperlong)         1682     1681
cor(Intercept:Sleepershort,Days:Sleepershort)       1851     2312

Regression Coefficients:
                 Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
Intercept          252.72     11.86   228.29   275.95 1.00     1880     2266
Days                14.58      2.17    10.25    18.90 1.00     1919     2213
Sleepershort        -3.66     16.79   -38.00    29.07 1.00     1898     2087
Days:Sleepershort   -8.28      2.93   -14.23    -2.45 1.00     2030     2125

Further Distributional Parameters:
      Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
sigma    26.05      1.59    23.14    29.31 1.00     3059     2780
```

Looking at the summary output, specifically, under "Multilevel Hyperparameters", we see only little differences between the standard deviations and correlations of the two `Sleeper` groups. However, the overall interaction effect of `Days` and `Sleeper` as shown under "Regression Coefficients" is quite strongly negative. That is, according to the results, shorter sleepers seems to be less affected by sleep deprivation than longer sleepers. Please keep in mind that I added the `Sleeper` variable artifically to showcase some of the functionality of brms on this data example. So this is not an actual empirical finding.

4.12 Some benefits of multilevel models

I want to close this chapter by a short discussion and summary on the benefits of multilevel models, both in general and specifically when applied in a Bayesian framework. I will start with the general points.

For me, perhaps the most important benefit of multilevel models is that they help us better calibrate the uncertainty of the overall coefficients in the face of structured data. This point

we have seen demonstrated among others in the small simulation study on Type 1 error rates. Further benefits include the convenient estimation of variation in different levels of the hierarchy, such that we can, for example, see how much variation in the data is attributable to variation between vs. within subjects. In scenarios where we care about the varying coefficients, additional benefits come into play such as their improved estimation accuracy through partially pooling information across groups. This also leads to better out-of-sample predictions especially in cases where the number of observations per group is small relative to the number of coefficients per group. Lastly, multilevel models allow us to easily predict the coefficients of new levels by using the posterior over the estimated varying coefficients as a prior.

In terms of Bayes-specific benefits – or at least benefits more often harvested in Bayesian model – I particularily see the greater modeling flexibility which tends to allow the estimation of more complex multilevel models including more grouping factors and more varying coefficients without running into serious convergence issues. This is driven by two main aspects: Firstly, specifying priors on the multilevel hyperparameters can help stabilizing estimation and obtain reasonable posteriors even in the face of sparse data or over-parameterized models where the data alone are insufficient to inform the parameters. Secondly, advanced sampling algorithms such as the MCMC methods implemented in Stan tend to be more powerful than optimization-based approaches in obtaining a good presentation of the posterior (or any form of frequentist equivalent). The use of sampling algorithms also enables the joint estimation of overall and varying coefficients, something that is much harder to achieve in optimization-based approaches. The price of sampling is usually an increase of estimation time by an order of magnitude or more compared to optimization. That said, there are algorithms that bridge the gap between optimization and sampling, including integrated laplace approximation (INLA) (Rue, Martino, and Chopin 2009), automatic differentiation variational inference (ADVI) (Kucukelbir et al. 2017), and pathfinder (Zhang et al. 2022). Specifally INLA is well suited for estimating Bayesian multilevel models both accuractly and efficiently and you should give it a try if your model is too slow to estimate in brms or Stan.

References

Bates, Douglas, Martin Mächler, Ben Bolker, and Steve Walker. 2015. "Fitting Linear Mixed-Effects Models Using \Pkglme4." *Journal of Statistical Software* 67 (1): 1–48.

Belenky, Gregory, 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). https://doi.org/10.1046/j.1365-2869.2003.00337.x.

Bernardo, José M., and Adrian F. M. Smith. 1994. *Bayesian Theory*. Hoboken: Wiley.

Bockting, Florence, Stefan T. Radev, and Paul-Christian Bürkner. 2023. "Simulation-Based Prior Knowledge Elicitation for Parametric Bayesian Models." *arXiv Preprint*. https://doi.org/10.48550/arXiv.2308.11672.

Breslow, N. E., and D. G. Clayton. 1993. "Approximate Inference in Generalized Linear Mixed Models." *Journal of the American Statistical Association* 88 (421): 9–25. https://doi.org/10.1080/01621459.1993.10594284.

Bürkner, Paul-Christian, Jonah Gabry, and Aki Vehtari. 2021. "Efficient Leave-One-Out Cross-Validation for Bayesian Non-Factorized Normal and Student-t Models." *Computational Statistics* 36 (2): 1243–61.

Bürkner, Paul-Christian, Maximilian Scholz, and Stefan T. Radev. 2023b. "Some Models Are Useful, but How Do We Know Which Ones? Towards a Unified Bayesian Model Taxonomy." *Statistics Surveys* 17 (none): 216–310. https://doi.org/10.1214/23-SS145.

———. 2023a. "Some Models Are Useful, but How Do We Know Which Ones? Towards a Unified Bayesian Model Taxonomy." *Statistics Surveys* 17: 216–310. https://doi.org/10.1214/23-SS145.

Gelman, Andrew, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. 2013. *Bayesian Data Analysis (3rd Edition)*. London: Chapman; Hall/CRC.

Gelman, Andrew, Ben Goodrich, Jonah Gabry, and Aki Vehtari. 2019. "R-Squared for Bayesian Regression Models." *The American Statistician* 73 (3): 307–9.

Gelman, Andrew, and Jennifer Hill. 2006. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.

Gelman, Andrew, Aki Vehtari, Daniel Simpson, Charles C Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. 2020. "Bayesian Workflow." *arXiv Preprint*.

Gronau, Quentin F., Alexandra Sarafoglou, Dora Matzke, Alexander Ly, Udo Boehm, Maarten Marsman, David S. Leslie, Jonathan J. Forster, Eric-Jan Wagenmakers, and Helen Steingroever. 2017. "A Tutorial on Bridge Sampling." *Journal of Mathematical Psychology* 81:

80–97. https://doi.org/10.1016/j.jmp.2017.09.005.

Gronau, Quentin F., Henrik Singmann, and Eric-Jan Wagenmakers. 2020. "Bridgesampling: An R Package for Estimating Normalizing Constants." *Journal of Statistical Software* 92 (10): 1–29. https://doi.org/10.18637/jss.v092.i10.

Hastie, Trevor, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Vol. 2. Springer.

Hendricks, Paul. 2015. *Titanic: Titanic Passenger Survival Data Set*. https://CRAN.R-project.org/package=titanic.

Juárez, Miguel A., and Mark F. J. Steel. 2010. "Model-Based Clustering of Non-Gaussian Panel Data Based on Skew- t Distributions." *Journal of Business & Economic Statistics* 28 (1): 52–66. https://doi.org/10.1198/jbes.2009.07145.

Kucukelbir, Alp, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. 2017. "Automatic Differentiation Variational Inference." *Journal of Machine Learning Research* 18 (14): 1–45.

Kurz, Solomon. 2019. "Statistical Rethinking with Brms, Ggplot2, and the Tidyverse." https://bookdown.org/ajkurz/Statistical_Rethinking_recoded/.

Lewandowski, Daniel, Dorota Kurowicka, and Harry Joe. 2009. "Generating Random Correlation Matrices Based on Vines and Extended Onion Method." *Journal of Multivariate Analysis* 100 (9): 1989–2001.

McCullagh, P. 2019. *Generalized Linear Models*. 2nd ed. New York: Routledge. https://doi.org/10.1201/9780203753736.

McElreath, Richard. 2019. *Statistical Rethinking: A Bayesian Course with Examples in R and Stan (2nd Edition)*. London: Chapman; Hall/CRC.

Meng, Xiao-Li, and Stephen Schilling. 2002. "Warp Bridge Sampling." *Journal of Computational and Graphical Statistics* 11 (3): 552–86. https://doi.org/10.1198/106186002457.

Mikkola, Petrus, Osvaldo A. Martin, Suyog Chandramouli, Marcelo Hartmann, Oriol Abril Pla, Owen Thomas, Henri Pesonen, et al. 2023. "Prior Knowledge Elicitation: The Past, Present, and Future." *Bayesian Analysis* -1: 1–33. https://doi.org/10.1214/23-BA1381.

Paananen, Topi, Juho Piironen, Paul-Christian Bürkner, and Aki Vehtari. 2021. "Implicitly Adaptive Importance Sampling." *Statistics and Computing* 31 (2). https://doi.org/10.1007/s11222-020-09982-2.

Rue, Håvard, Sara Martino, and Nicolas Chopin. 2009. "Approximate Bayesian Inference for Latent Gaussian Models by Using Integrated Nested Laplace Approximations." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 71 (2): 319–92.

Säilynoja, Teemu, Paul-Christian Bürkner, and Aki Vehtari. 2022. "Graphical Test for Discrete Uniformity and Its Applications in Goodness-of-Fit Evaluation and Multiple Sample Comparison." *Statistics and Computing* 32 (2). https://doi.org/10.1007/s11222-022-10090-6.

Scholz, Maximilian, and Paul-Christian Bürkner. 2023. "Prediction Can Be Safely Used as a Proxy for Explanation in Causally Consistent Bayesian Generalized Linear Models." *arXiv Preprint*. https://doi.org/10.48550/arXiv.2210.06927.

Sivula, Tuomas, Måns Magnusson, Asael Alonzo Matamoros, and Aki Vehtari. 2022. "Uncertainty in Bayesian Leave-One-Out Cross-Validation Based Model Comparison." *arXiv*

Preprint.

Thall, Peter F, and Stephen C Vail. 1990. "Some Covariance Models for Longitudinal Count Data with Overdispersion." *Biometrics*, 657–71.

Vehtari, Aki, Andrew Gelman, Daniel Simpson, Bob Carpenter, and Paul-Christian Bürkner. 2021. "Rank-Normalization, Folding, and Localization: An Improved $\widehatR$ for Assessing Convergence of MCMC (with Discussion)." *Bayesian Analysis* 16 (2): 667–718.

Vehtari, Aki, and Janne Ojanen. 2012. "A Survey of Bayesian Predictive Methods for Model Assessment, Selection and Comparison." *Statistics Surveys* 6: 142–228.

Vehtari, Aki, Daniel Simpson, Andrew Gelman, Yuling Yao, and Jonah Gabry. 2024. "Pareto Smoothed Importance Sampling." *Journal of Machine Learning Research* 25 (72): 1–58. http://jmlr.org/papers/v25/19-556.html.

Wagenmakers, Eric-Jan, and Simon Farrell. 2004. "AIC Model Selection Using Akaike Weights." *Psychonomic Bulletin & Review* 11 (1): 192–96. https://doi.org/10.3758/BF03206482.

Yao, Yuling, Aki Vehtari, Daniel Simpson, and Andrew Gelman. 2018. "Using Stacking to Average Bayesian Predictive Distributions (with Discussion)." *Bayesian Analysis* 13 (3): 917–1007. https://doi.org/10.1214/17-BA1091.

Zhang, Lu, Bob Carpenter, Andrew Gelman, and Aki Vehtari. 2022. "Pathfinder: Parallel Quasi-Newton Variational Inference." *Journal of Machine Learning Research* 23 (306): 1–49.