More Examples of Model Formula Syntax

Bayesian Data Analysis

Steve Buyske

Coding a wider range of models

- At this point, it's worth spending a little time on a wider range of linear models and the accompany syntax.
- So far, we've talked about
  - non-hierarchical regression models, with an R syntax for the formula like `y ~ u * v + x`.
  - hierarchical regression models, with an R syntax for the formula such as `y ~ u * v + x + (x | w)`.
- In this segment, we'll look a little at more complicated hierarchical models, at a simple comparison of two groups, at different package for Bayesian model fitting, and at "robust regression".

Hierarchical models with crossed grouping factors

- We may have situations where observations are grouped in more than one way.
  - For the SAT and ACT exams, if you think of an observation as one student's answer to one test item, then observations are grouped by
    - student
    - item
- If you've know some experimental design, you might think of a factorial random effects experiment.

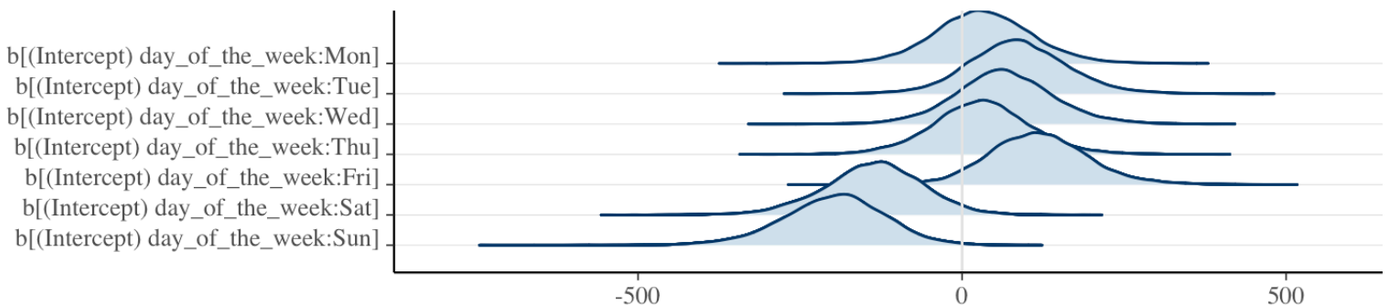
- Think back to the citibike example.
- We fit a linear trend for day of the year as well as a day-of-the-week effect, using `rides ~ day + (1 | day_of_the_week)`
- What if instead we wanted to drop the day of the year trend, but add a week-of-the-year effect as well as a day-of-the-week effect.
    - That is, we think of each observation as belonging to a week of the year, as well as to a day of the week.
- In the experimental design literature these are known as *crossed* factors.

- The R syntax for crossed factors is straightforward: you just add an additional term like `(1 | group)`.
- For example, for the citibike data, our new formula would be
  - `rides ~  (1 | day_of_the_week) + (1 | week)`
    - (Notice I dropped `day`, but I could have potentially included it with
    - `rides ~ days +  (1 | day_of_the_week) + (1 | week)`
- Although the code is simple to write, in part because there are more terms in the model it may require more iterations to fit.

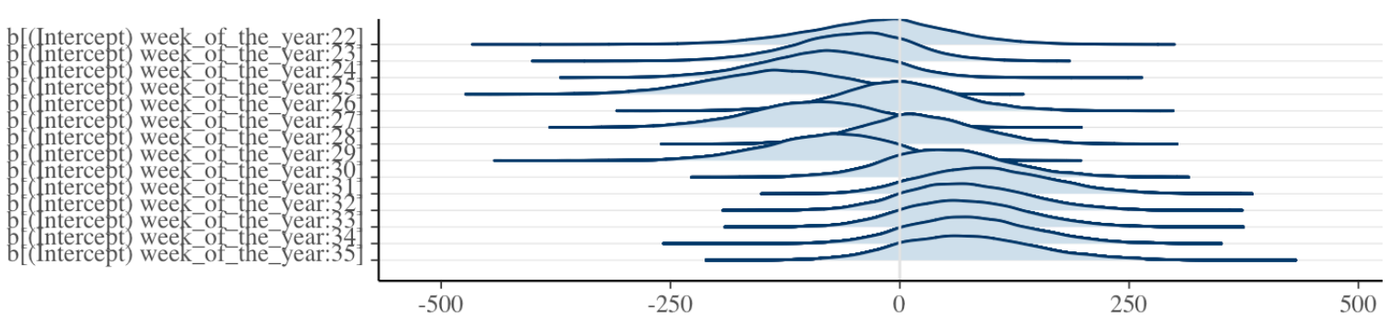
```
citibike_crossed <- citibike2 %>% filter(year == "2019") %>%
    stan_glmer(rides ~  (1 | day_of_the_week) + (1 | week_of_the_year), data = ., iter = 4000, chains = 9, cores = 9)
```

The posteriors for the day-of-the-week effects look pretty similar to what we had before, while the week-of-the-year posteriors show some pretty clear effects:

```
plot(citibike_crossed, plotfun = "areas_ridges", regex_pars = "Intercept.+day.of.the.week")
```



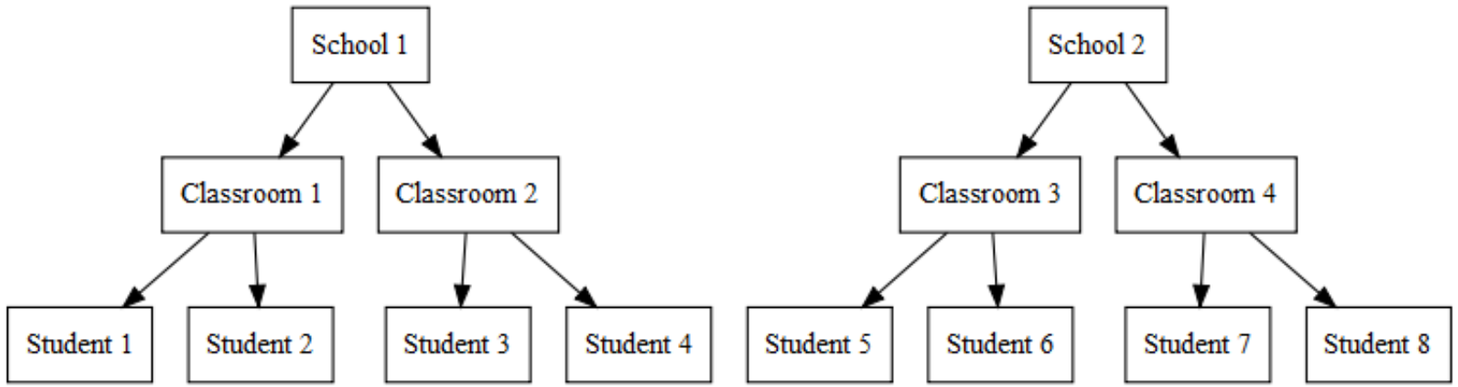

```
plot(citibike_crossed, plotfun = "areas_ridges", regex_pars = "Intercept.+week.of.the.year")
```



Hierarchical models with nested grouping factors

- Data is often organized with more than one level in the hierarchy. For example,
  - students grouped by their teacher and teachers organized by school (and maybe schools organized by town, state, or region)
  - lab measurements grouped by clinical trial participant, with participant grouped by study center.

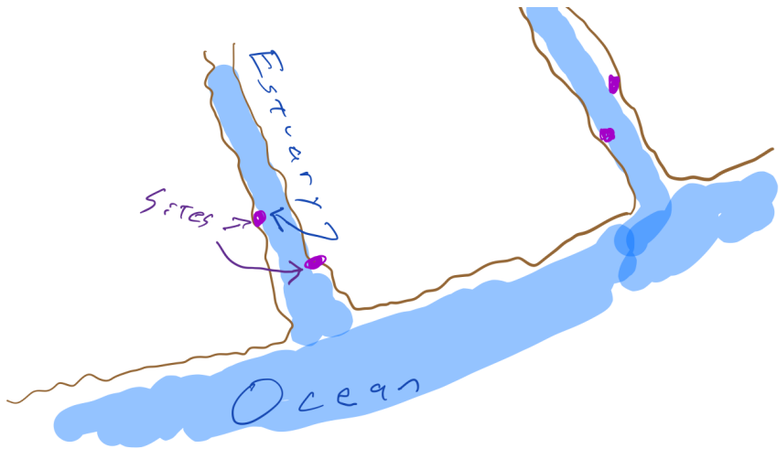
- If each group has a unique id, then you can use the same syntax as for crossed groups
  - but if, say, teacher A might be two different people, depending on the school, then the syntax is more complicated, since you need to indicate that teacher A in school 1 is different from teacher A in school 2.



- Let's look at an environmental example, taken from http://environmentalcomputing.net/mixed-models-2/.

the study "aimed to test the effect of water pollution on the abundance of some subtidal marine invertebrates by comparing samples from modified and pristine estuaries"

- We will use `log(Total)` as the outcome.
- For each estuary (tidal mouth of river), there were several collection sites
  - Observations are nested in sites, which are nested in estuaries.



```
estuaries %>%
  glimpse()
```

```
## Rows: 54
## Columns: 8
## $ X1                    <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 1…
## $ Modification          <chr> "Modified", "Modified", "Modified", "Modified", …
## $ Estuary               <chr> "JAK", "JAK", "JAK", "JAK", "JAK", "JAK", "JAK",…
## $ Site                  <fct> 1, 1, 2, 2, 3, 3, 4, 4, 1, 1, 2, 2, 3, 3, 4, 4, …
## $ Hydroid               <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 7, 5, 2, 0, 0, 0, 3, 3, …
## $ Total                 <dbl> 44, 42, 32, 44, 42, 48, 45, 34, 29, 51, 12, 29, …
## $ Schizoporella.errata <dbl> 15, 8, 9, 14, 6, 12, 28, 1, 0, 0, 0, 0, 0, 0, 0,…
## $ SiteWithin            <fct> JAK:1, JAK:1, JAK:2, JAK:2, JAK:3, JAK:3, JAK:4,…
```

We fit this with the following model. You can ignore `prior_covariance = decov(shape = 5)`; we will return to that next week.

```
estuaries_hier <- stan_glmer(
  log(Total) ~ Modification + (1 | Estuary) + (1 | SiteWithin),
  data = estuaries,
  iter = 4000,
  chains = 7,
  cores = 7,
  prior_covariance = decov(shape = 5)
)
```

- If you have a variable encoded in a way that's not unique, like teacher A in school 1 and teacher A in school B, you can change the model formula to something like (note the `:`)
- `log(Total) ~ Modification + (1 | Estuary) + (1 | Site:Estuary)`

Estuary results

```
estuaries_hier %>%
  as.data.frame() %$%
  mean(ModificationPristine < 0)
```

```
## [1] 0.8718571
```

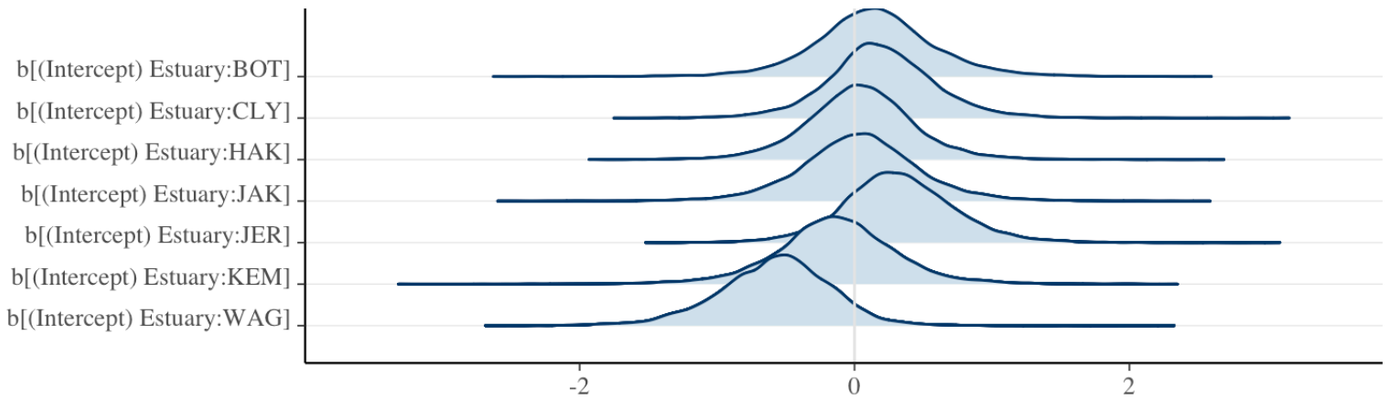
```
model_parameters(estuaries_hier, centrality = "Mean", ci = .9)
```

```
## # Fixed effects
##
## Parameter            |  Mean |        90% CI |     pd | % in ROPE |  Rhat |     ESS |                 Prior
## ------------------------------------------------------------------------------------------------------------
## (Intercept)          |  3.67 | [ 3.02, 4.40] |   100% |     0.01% | 1.000 | 8426.36 | Normal (3.34 +- 1.66)
## ModificationPristine | -0.58 | [-1.48, 0.32] | 87.19% |    92.01% | 1.000 | 7791.21 | Normal (0.00 +- 3.35)
```

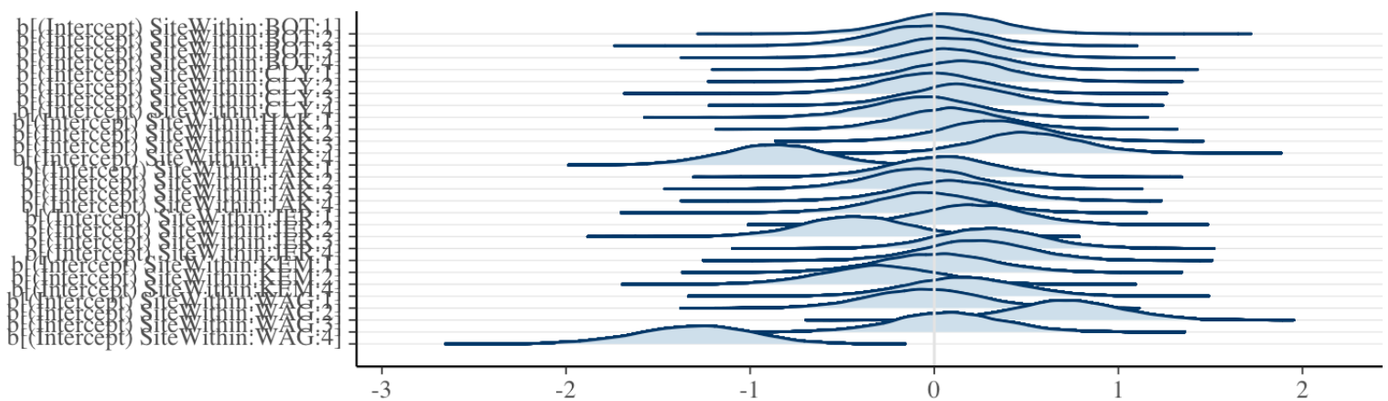
```
exp(-0.58)
```

```
## [1] 0.5598984
```

```
plot(estuaries_hier, plotfun = "areas_ridges", regex_pars = "Intercept.+Estuary")
```




```
plot(estuaries_hier, plotfun = "areas_ridges", regex_pars = "Intercept.+Site")
```



- Notice how much variability from site to site for the WAG estuary.
  - That would lead me to talk to the biologist about what might be going on with the data.

Two last models.

- These last two models are really different parametrization of different model.
- As a first motivating example, suppose we wanted to model the citibike data without including anything but year.

```
citibike_simple <- stan_glm(rides ~ year, data = citibike, iter = 4000)
```

```
model_parameters(citibike_simple)
```

```
## # Fixed effects
##
## Parameter   |  Median |            89% CI |     pd | % in ROPE |  Rhat |     ESS |                      Prior
## ------------------------------------------------------------------------------------------------------------------
## (Intercept) | 1433.80 | [1389.81, 1479.03] |   100% |        0% | 1.000 | 9051.05 | Normal (1362.80 +- 689.17)
## year2020    | -141.24 | [-202.61,  -79.84] | 99.99% |     0.22% | 1.000 | 8576.87 |   Normal (0.00 +- 1374.59)
```

We might want to focus on the values for each group, rather than a parameterization of [2019 mean] + [year effect]. You can get that by getting rid of the intercept in the model, which you do with `-1` in the formula.

```
citibike_simple_alt <- stan_glm(rides ~ -1 + year, data = citibike, iter = 4000)
```

```
model_parameters(citibike_simple_alt)
```

```
## # Fixed effects
##
## Parameter |  Median |            89% CI |   pd | % in ROPE |  Rhat |     ESS |                 Prior
## ------------------------------------------------------------------------------------------------
## year2019  | 1432.98 | [1390.10, 1478.18] | 100% |        0% | 1.000 | 7365.19 | Normal (0 +- 1374.59)
## year2020  | 1291.38 | [1246.92, 1337.13] | 100% |        0% | 1.000 | 8186.76 | Normal (0 +- 1374.59)
```

The models are the same, it's just that the parametrization differs. The same question (phrased appropriated for the parametrization) will give the same answer, up to differences due to the second MC in MCMC.

```
citibike_simple %>%
  as.data.frame() %$%
  mean(year2020 < 0)
```

```
## [1] 0.999875
```

```
citibike_simple_alt %>%
  as.data.frame() %$%
  mean(year2020 < year2019)
```

```
## [1] 0.999375
```

- As a second motivating example, let's go back to the exam example that we fit with
  - `stan_glmer(formula = normexam ~ standLRT + (standLRT | school), data = Exam)`
- Suppose we really just wanted the school-specific regression coefficent of standLRT, with no interest in the overall estimate.
  - In that case, we could drop the first `standLRT` and just keep the term inside the group part.
  - `stan_glmer(formula = normexam ~  (standLRT | school), data = Exam)`

```
exam_hier_alt <- stan_glmer(formula = normexam ~  (standLRT | school),  data = Exam, iter = 4000, chains = 16, cores = 16)
```

Notice that the only overall parameter is the intercept:

```
model_parameters(exam_hier_alt )
```

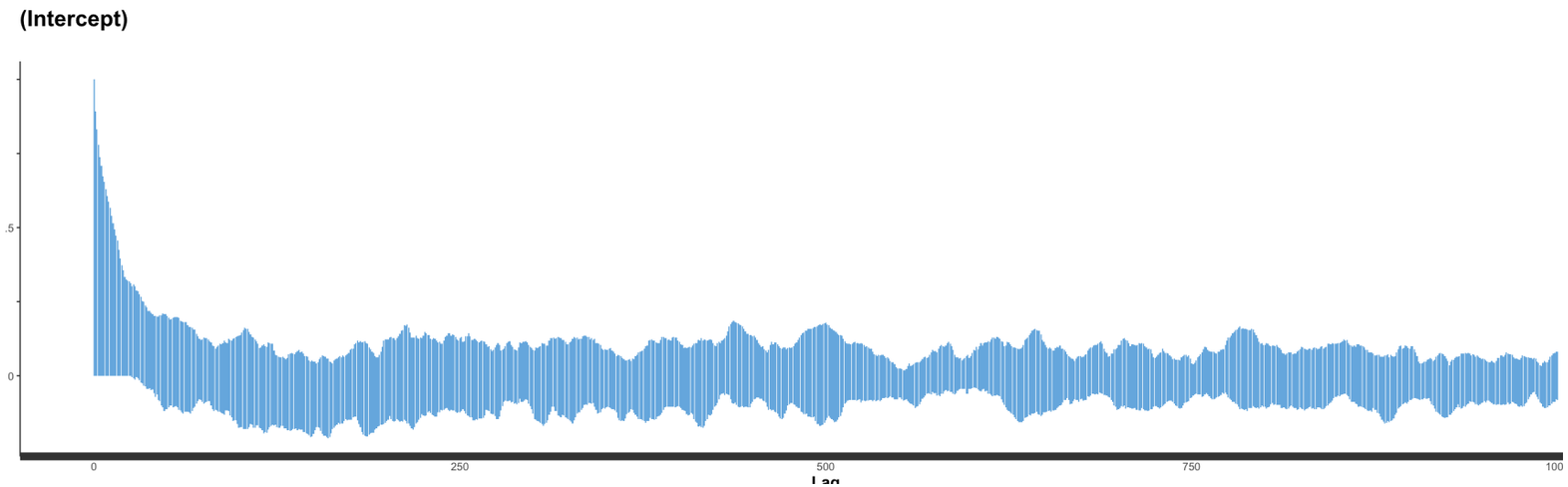
```
## # Fixed effects
##
## Parameter   | Median |         89% CI |     pd | % in ROPE |  Rhat |     ESS |                      Prior
## ------------------------------------------------------------------------------------------------------
## (Intercept) |  -0.31 | [-0.50, -0.10] | 99.43% |     4.50% | 1.010 | 1178.42 | Normal (-1.14e-04 +- 2.50)
```

- Notice that the school-specific slopes don't vary around 0, but instead vary around 0.55:

```
model_parameters(exam_hier_alt, effects = "random", centrality = "Mean", ci = .9, rope_ci = .9)
```

```
## # Fixed effects
##
## Parameter   | Mean | 90% CI | pd | % in ROPE | Rhat | ESS |                     Prior
## ------------------------------------------------------------------------------------
## (Intercept) |      |        |    |           |      |     | Normal (-1.14e-04 +- 2.50)
##
## # Random effects Intercept: school
##
## Parameter |     Mean |         90% CI |     pd | % in ROPE |  Rhat |      ESS |     Prior
## ------------------------------------------------------------------------------------------
## school:1  |     0.67 | [ 0.42,  0.92] |   100% |        0% | 1.006 |  1714.02 | NA ( +- )
## school:2  |     0.75 | [ 0.49,  1.01] |   100% |        0% | 1.006 |  1881.54 | NA ( +- )
## school:3  |     0.78 | [ 0.52,  1.04] |   100% |        0% | 1.005 |  2111.57 | NA ( +- )
## school:4  |     0.32 | [ 0.08,  0.58] | 98.42% |     2.00% | 1.007 |  1608.87 | NA ( +- )
## school:5  |     0.54 | [ 0.27,  0.83] | 99.93% |        0% | 1.005 |  2176.71 | NA ( +- )
```

- Models fit with group effects but not overall effects may be *very* slow to converge. Here's the autocorrelation plot of the model, out to a lag of *1,000*.



- Even with 32,000 draws from the posterior, the effective sample size for some of the parameters is only about 1,200 because there's so much autocorrelation.

A different package to fit Bayesian models.

- Although we are working in R, the Bayesian model fitting is actually done outside of R.
- There is a specialized language called Stan (named in honour of Stanislaw Ulam, pioneer of the Monte Carlo method) that does the fitting (and itself generates C++ code).
- It is possible to run Stan code explicitly from R, using a package called `rstan`; we won't do it that way.
- `stan_glm()` and `stan_glmer()` are functions in the `rstanarm` package
  - The `rstanarm` package is more or less straightforward to use, but the models are pre-compiled.
  - That means that you can only fit models that have been pre-programmed in.

The `brms` package

- The `brms` package uses a similar syntax, but compiles the code *after* you call the function.
  - There will always be a lag of a minute or two while the code is compiled.
    - You'll see a message of `Compiling Stan program...`
  - There is a lot more flexibility, however.
- We will start with a new example about performance on a cognitive exam for two groups on different medications.

```
test %>% glimpse()
```

```
## Rows: 40
## Columns: 2
## $ score <dbl> 111, 110, 110, 95, 125, 109, 120, 141, 116, 111, 114, 122, 83, …
## $ group <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A"…
```

Let's just fit a very simple model of score as the outcome and group as the predictor. First using `stan_glm()`.

```
test_rstanarm <- stan_glm(score ~ group, data = test, iter = 5000)
```

The main function in `brms` is `brm()`

```
library(brms)

test_brms <- brm(score ~ group, data = test, iter = 5000)
```

- If you look at `model_parameters()`, you'll see that the posteriors are a little different.

```
model_parameters(test_rstanarm, ci = .9)
```

```
## # Fixed effects
##
## Parameter   | Median |           90% CI |     pd | % in ROPE |  Rhat |     ESS |                   Prior
## ------------------------------------------------------------------------------------------------------
## (Intercept) | 108.01 | [ 98.62, 116.71] |   100% |        0% | 1.000 | 7749.61 | Normal (103.95 +- 61.17)
## groupB      |  -8.28 | [-21.61,   4.50] | 85.52% |    13.46% | 1.000 | 7700.18 |  Normal (0.00 +- 120.81)
```

```
model_parameters(test_brms, ci = .9)
```

```
## # Fixed effects
##
## Parameter   | Median |           90% CI |     pd | % in ROPE |  Rhat |     ESS
## ------------------------------------------------------------------------------
## (Intercept) | 108.16 | [ 99.23, 117.06] |   100% |        0% | 1.000 | 8357.41
```

- Part of that is due to chance, but the biggest reason is that the default priors for the two packages are different. The `brms` uses uniform priors, while `rstanarm` uses (wide) normals. More on this later.

- Another difference is that the `summary()` function for an object created by `brm()` is different from that for an object created by `stan_glm()` and `stan_glmer()`. I think it's more useful.

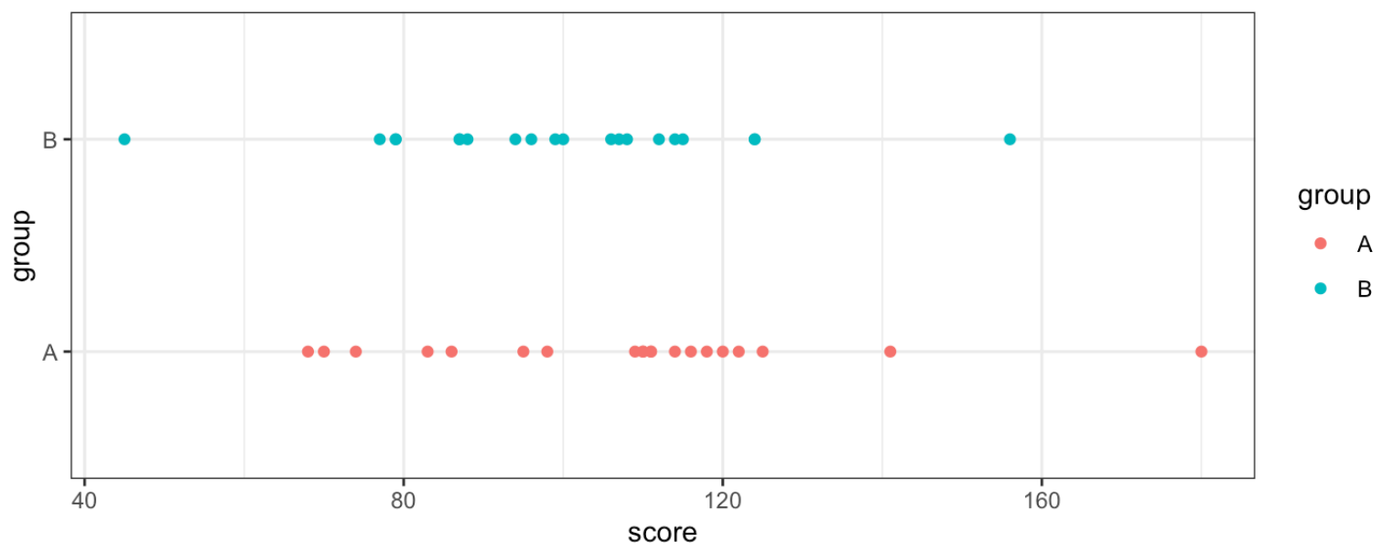
```
summary(test_brms)
```

```
##  Family: gaussian
##   Links: mu = identity; sigma = identity
## Formula: score ~ group
##    Data: test (Number of observations: 40)
## Samples: 4 chains, each with iter = 5000; warmup = 2500; thin = 1;
##          total post-warmup samples = 10000
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept   108.22      5.49    97.37   118.99 1.00     8413     7071
## groupB       -8.18      7.80   -23.56     7.21 1.00     8301     6557
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
```

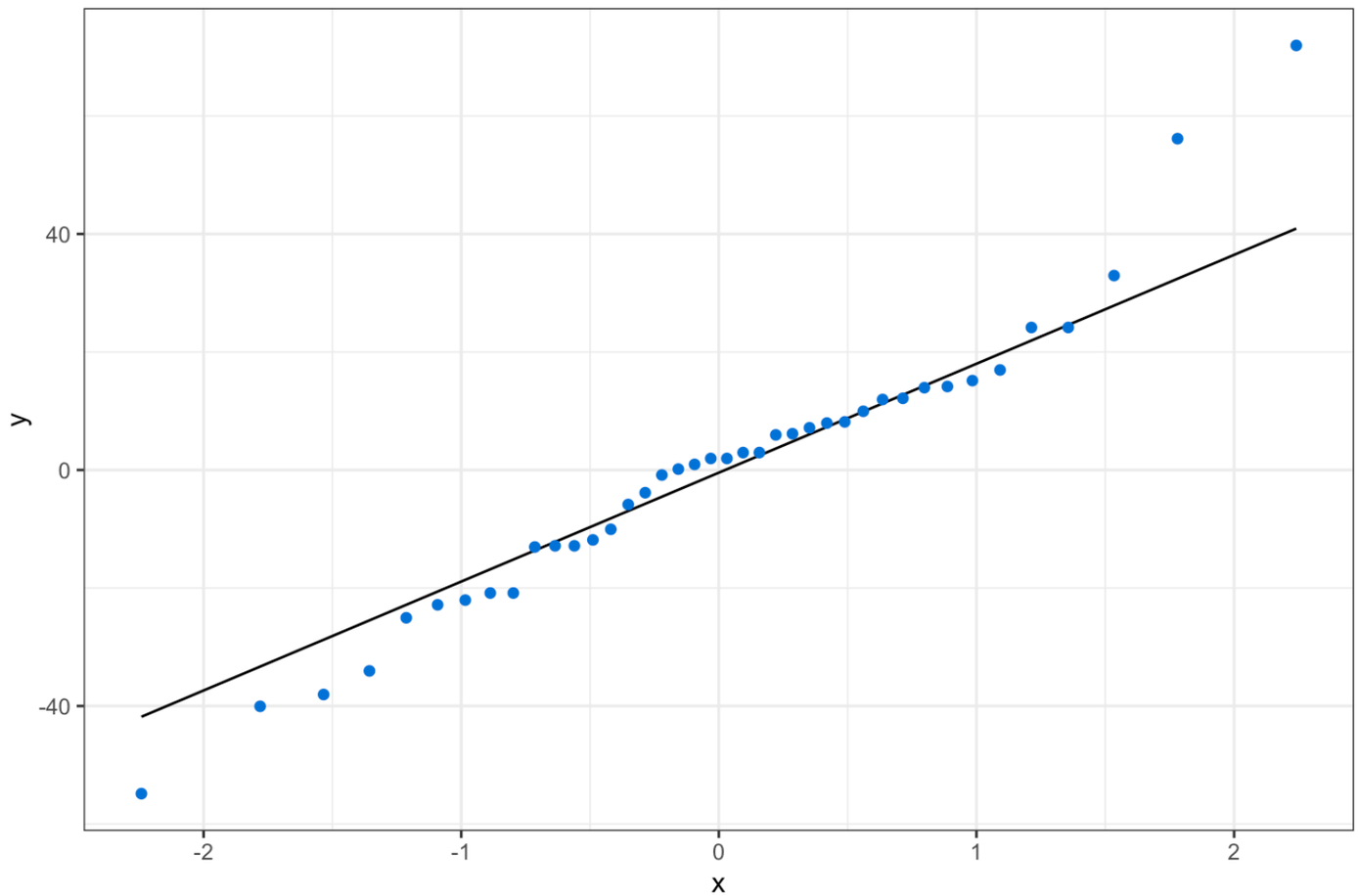
Robust regression

- Our models so far have all involved something like $y_i \sim N(\mu_i, \sigma)$.
- Consider a simple example of two groups, where there seem to be some values are father out than you might expect.

```
test %>%
  ggplot(aes(score, group, colour = group)) +
  geom_point() +
  theme_bw()
```



- You can see it in the QQ plot.



- Instead of using $y_i \sim N(\mu_i, \sigma)$, we might use a t-distribution, since it allows for heavier tails.
  - That is, more probability farther away.

- You can see here the difference in how a likelihood based on a normal and on a t-distribution fit points with an outlier.
- The normal is the blue one that's spread out.

Fitting a robust regression

- With the `brm()` function, to change from normal to Student's t distribution, you only need to change the `family` argument.

```
test_brms_t <- brm(score ~ group, data = test, iter = 5000, family = student)
```

```
model_parameters(test_brms, ci = .9)
```

```
## # Fixed effects
##
## Parameter   | Median |           90% CI |     pd | % in ROPE |  Rhat |     ESS
## ------------------------------------------------------------------------------
## (Intercept) | 108.16 | [ 99.23, 117.06] |   100% |        0% | 1.000 | 8357.41
## groupB      |  -8.17 | [-21.26,   4.36] | 85.89% |    14.09% | 1.000 | 8218.27
```

```
model_parameters(test_brms_t, ci = .9)
```

```
## # Fixed effects
##
## Parameter   | Median |           90% CI |     pd | % in ROPE |  Rhat |     ESS
## ------------------------------------------------------------------------------
## (Intercept) | 107.59 | [ 99.43, 116.66] |   100% |        0% | 1.000 | 8688.44
## groupB      |  -7.55 | [-20.06,   4.07] | 85.44% |    15.51% | 1.000 | 9269.71
```

Kruschke diagram for robust regression