Bayesian Regression using `brms`[1]

Mark Andrews
Psychology Department, Nottingham Trent University

@xmjandrews
mark.andrews@ntu.ac.uk
https://github.com/mark-andrews/bps-cogdev19-brms

[1] These slides are not intended to be self-contained and comprehensive, but just aim to provide some of the workshop's content. Elaborations and explanations will be provided in the workshop itself.

Introduction

- The R package `brms` is a easy to use but powerful tool for doing Bayesian regression.
- It allows us to do general and generalized linear models and their multilevel counterparts, almost as easily as with `lm`, `glm`, `lmer`, etc.
- It includes far more probability models for outcome variables than almost all other regression packages: gaussian, student, binomial, bernoulli, poisson, negbinomial, geometric, Gamma, skew_normal, lognormal, shifted_lognormal, exgaussian, wiener, inverse.gaussian, exponential, weibull, frechet, Beta, von_mises, asym_laplace, gen_extreme_value, categorical, cumulative, cratio, sratio, acat, hurdle_poisson, hurdle_negbinomial, hurdle_gamma, hurdle_lognormal, zero_inflated_binomial, zero_inflated_beta, zero_inflated_negbinomial, zero_inflated_poisson, and zero_one_inflated_beta.
- It also allows for censored data, missing data, measurment error, nonlinear regression, probabilistic mixture models, *distributional* models (whereby all parameters of the outcome variables have predictors), and so on.

Disclaimer

There are some major topics that we can not cover in depth:

- The nature of Bayesian data analysis
- The what, why, and how of Markov Chain Monte Carlo
- The what, why, and how of probabilistic programming languages

The how and why of Brms

- Brms writes Stan and Stan writes and compiles a MCMC sampler.
- To understand this process and its importance better, we must appreciate the following:
  1. Bayes is best. No further discussion necessary.
  2. Doing Bayesian data analysis, except for when using a prohibitively small set of models, requires Markov Chain Monte Carlo (MCMC) samplers.
  3. Writing your own MCMC is either hard or very hard.
  4. Probabilistic programming languages like Stan essentially write your MCMC sampler for any model you programmatically define.
  5. Although probabilistic programming languages reduce down the time and effort to obtain your sampler by orders of magnitude, they *still* require considerable time and effort relative to writing a single R command.
- Brms allows you to write your Bayesian model (with some restrictions) using standard R regression commands.

Load packages and data

```
> library(tidyverse)
> library(brms)
> library(modelr)
> library(lme4)
> library(magrittr)
>
> theme_set(theme_classic())
>
> # data
> weight_df <- read_csv('data/weight.csv')
> insul_df <- read_csv('data/insulation.csv')
> titanic_df <- read_csv('data/titanic.csv')
> sleep_df <- read_csv('data/sleepstudy.csv')
> science_df <- read_csv('data/science.csv')
>
> options(mc.cores = 2)
> #options(mc.cores = parallel::detectCores())
```

Simple linear regression

```
> # classic
> M_lm <- lm(weight ~ height, data = weight_df)
>
> # Bayesian
> M_brm <- brm(weight ~ height, data = weight_df)
```

Simple linear regression (cont'd)

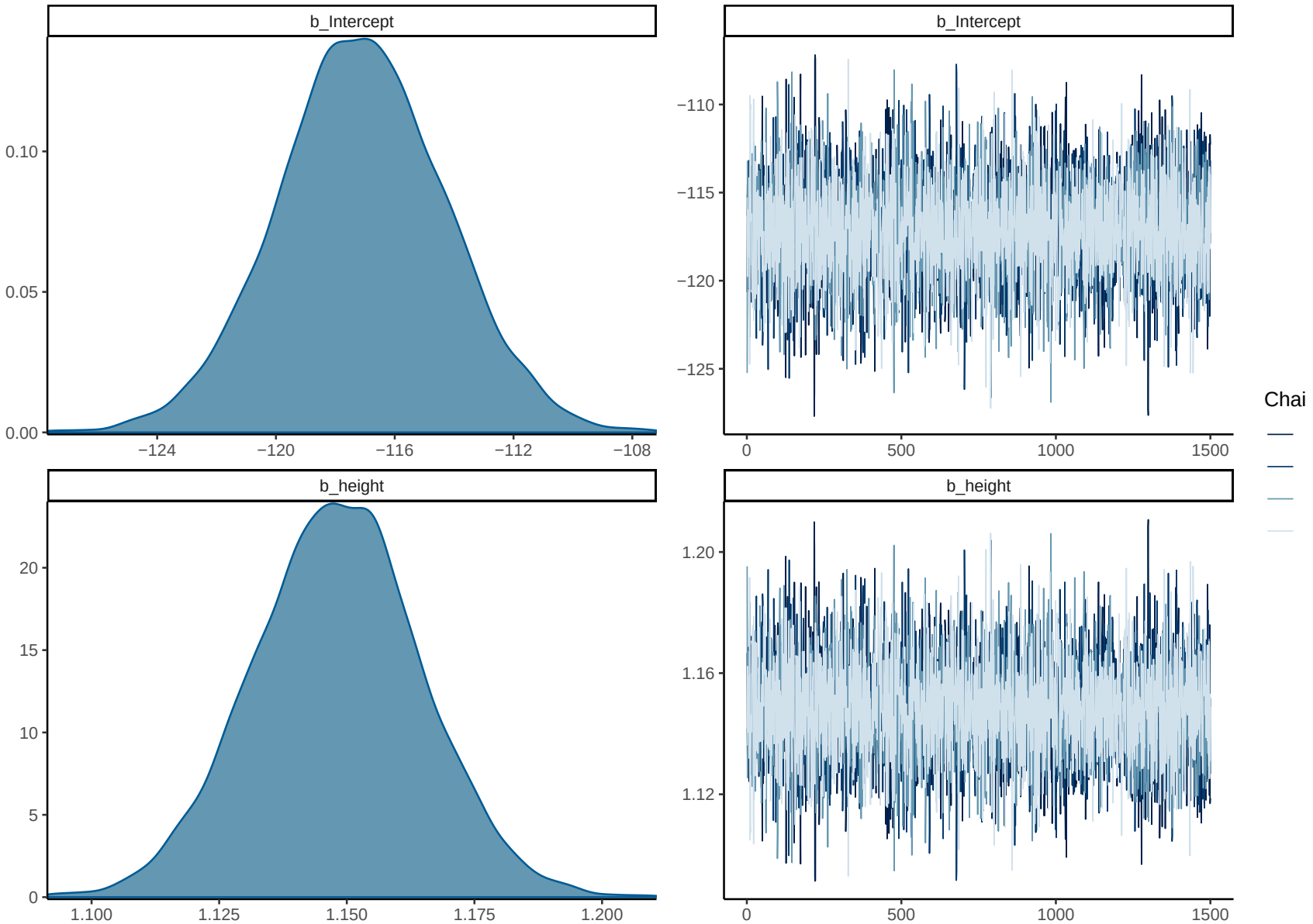
```
> summary(M_brm)
#>  Family: gaussian
#>   Links: mu = identity; sigma = identity
#> Formula: weight ~ height
#>    Data: weight_df (Number of observations: 6068)
#> Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
#>          total post-warmup samples = 4000
#>
#> Population-Level Effects:
#>           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> Intercept  -117.14      2.91  -122.86  -111.40 1.00     3552     2716
#> height        1.15      0.02     1.11     1.18 1.00     3558     2793
#>
#> Family Specific Parameters:
#>       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
#> sigma    11.76      0.11    11.55    11.98 1.00     4002     2617
#>
#> Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
#> is a crude measure of effective sample size, and Rhat is the potential
#> scale reduction factor on split chains (at convergence, Rhat = 1).
```

Simple linear regression (cont'd)

```
> # Overriding defaults
> M_brm <- brm(weight ~ height,
+              data = weight_df,
+              cores = 2, # I have a dual-core
+              chains = 4, # 4 chains is typical
+              iter = 2500,
+              warmup = 1000, # initilization etc
+              # flat(ish) prior on coefs
+              prior = set_prior('normal(0, 100)'),
+              seed = 101011 # for reproducibility
+ )
```

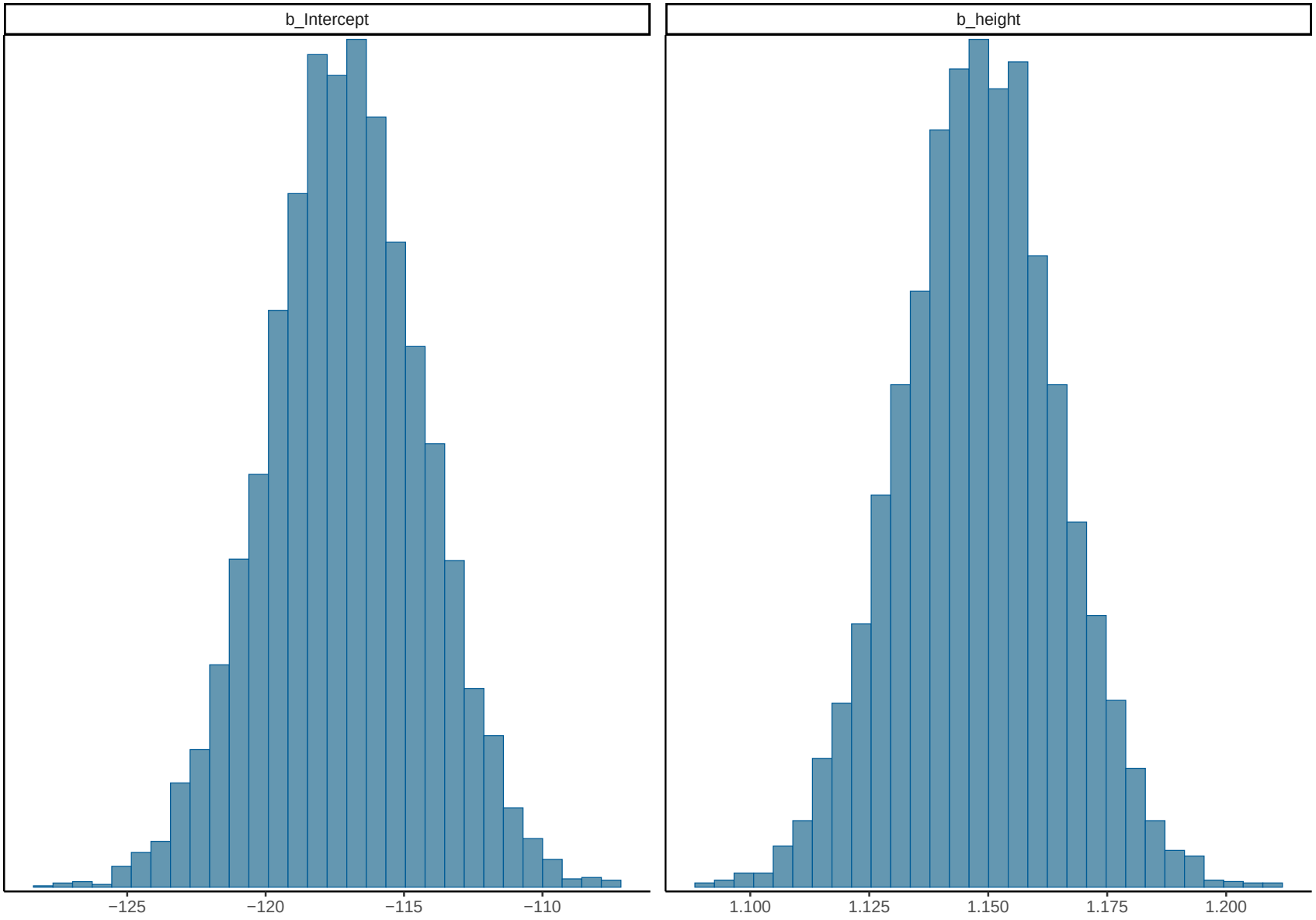
Plot the posterior distributions

```
> # plot just coefficients
> plot(M_brm, pars = '^b')
```



Plot posterior intervals

```
> stanplot(M_brm, type='hist', pars='^b')
```



Posterior predictive checks

Marginal plots

```
> marginal_effects(M_brm)
```

Posterior samples

```
> posterior_samples(M_brm)
```

Get predictions

```
> predict(M_brm)
>
> # predictions with new data
> tibble(height = c(160, 170, 180)) %>%
+   add_predictions(M_brm)
```

Get information on priors

```
> prior_summary(M_brm)
#>                   prior     class   coef group resp dpar nlpar bound
#> 1       normal(0, 100)         b
#> 2                              b height
#> 3 student_t(3, 78, 16) Intercept
#> 4  student_t(3, 0, 16)     sigma
```

View the stan code

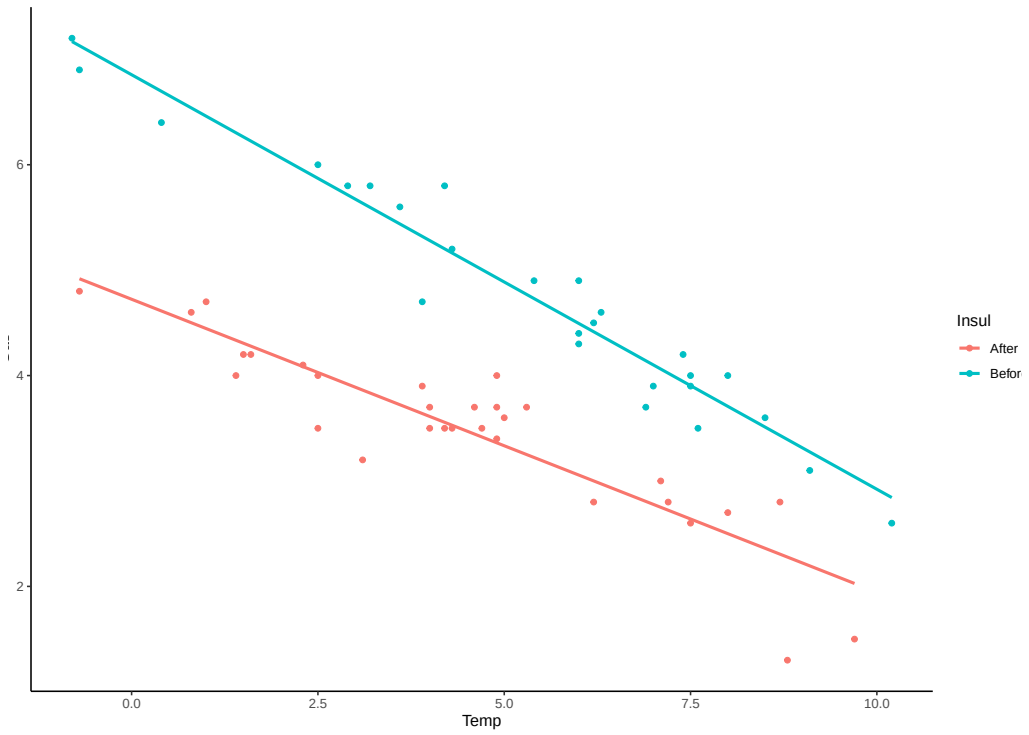
```
> stancode(M_brm)
```

Change priors

```
> # Change priors
> newpriors <- c(prior_string("student_t(3, 0, 10)", class = "b"
+                prior_string("student_t(3, 18, 10)", class = "I
+                prior_string("student_t(3, 0, 10)", class = "si
>
> M_brm <- brm(weight ~ height,
+              data = weight_df,
+              cores = 2,
+              chains = 4,
+              iter = 2500,
+              warmup = 1000,
+              prior = newpriors,
+              seed = 101011
+ )
```

Model comparison

```
> ggplot(insul_df,
+        mapping = aes(x = Temp, y = Gas, col = Insul)
+ ) + geom_point() + stat_smooth(method = 'lm', se = F)
```



Interaction linear model

```
> M_lm <- lm(Gas ~ Temp*Insul, data=insul_df)
>
> M_bayes <- brm(Gas ~ Temp*Insul,
+                data = insul_df,
+                cores = 2,
+                prior = set_prior('normal(0, 100)'),
+                save_all_pars = T
+ )
```

Additive model

```
> # We'll do a model comparison comparing the above
> # model to an additive, i.e. non-interaction, model
>
> M_lm_additive <- lm(Gas ~ Temp+Insul, data = insul_df)
> M_bayes_additive <- brm(Gas ~ Temp+Insul,
+                         data = insul_df,
+                         cores = 2,
+                         prior = set_prior('normal(0, 100)'),
+                         save_all_pars = T
+ )
```

Compare additive and interaction models (waic)

```
> waic(M_bayes_additive, M_bayes)
#> Output of model 'M_bayes_additive':
#>
#> Computed from 4000 by 56 log-likelihood matrix
#>
#>           Estimate  SE
#> elpd_waic    -24.5 4.8
#> p_waic         3.8 0.7
#> waic          49.0 9.5
#>
#> Output of model 'M_bayes':
#>
#> Computed from 4000 by 56 log-likelihood matrix
#>
#>           Estimate   SE
#> elpd_waic    -19.8  6.7
#> p_waic         5.4  1.8
#> waic          39.6 13.3
#>
#> Model comparisons:
```

Compare additive and interaction models (looic)

```
> loo(M_bayes_additive, M_bayes)
#> Output of model 'M_bayes_additive':
#>
#> Computed from 4000 by 56 log-likelihood matrix
#>
#>          Estimate  SE
#> elpd_loo    -24.5 4.8
#> p_loo         3.8 0.7
#> looic        49.0 9.5
#> ------
#> Monte Carlo SE of elpd_loo is 0.0.
#>
#> All Pareto k estimates are good (k < 0.5).
#> See help('pareto-k-diagnostic') for details.
#>
#> Output of model 'M_bayes':
#>
#> Computed from 4000 by 56 log-likelihood matrix
#>
#>          Estimate   SE
```

Compare additive and interaction models (Bayes factor)

```
> bayes_factor(M_bayes_additive, M_bayes)
#> Iteration: 1
#> Iteration: 2
#> Iteration: 3
#> Iteration: 4
#> Iteration: 5
#> Iteration: 1
#> Iteration: 2
#> Iteration: 3
#> Iteration: 4
#> Iteration: 5
#> Estimated Bayes factor in favor of bridge1 over bridge2: 9.75
```

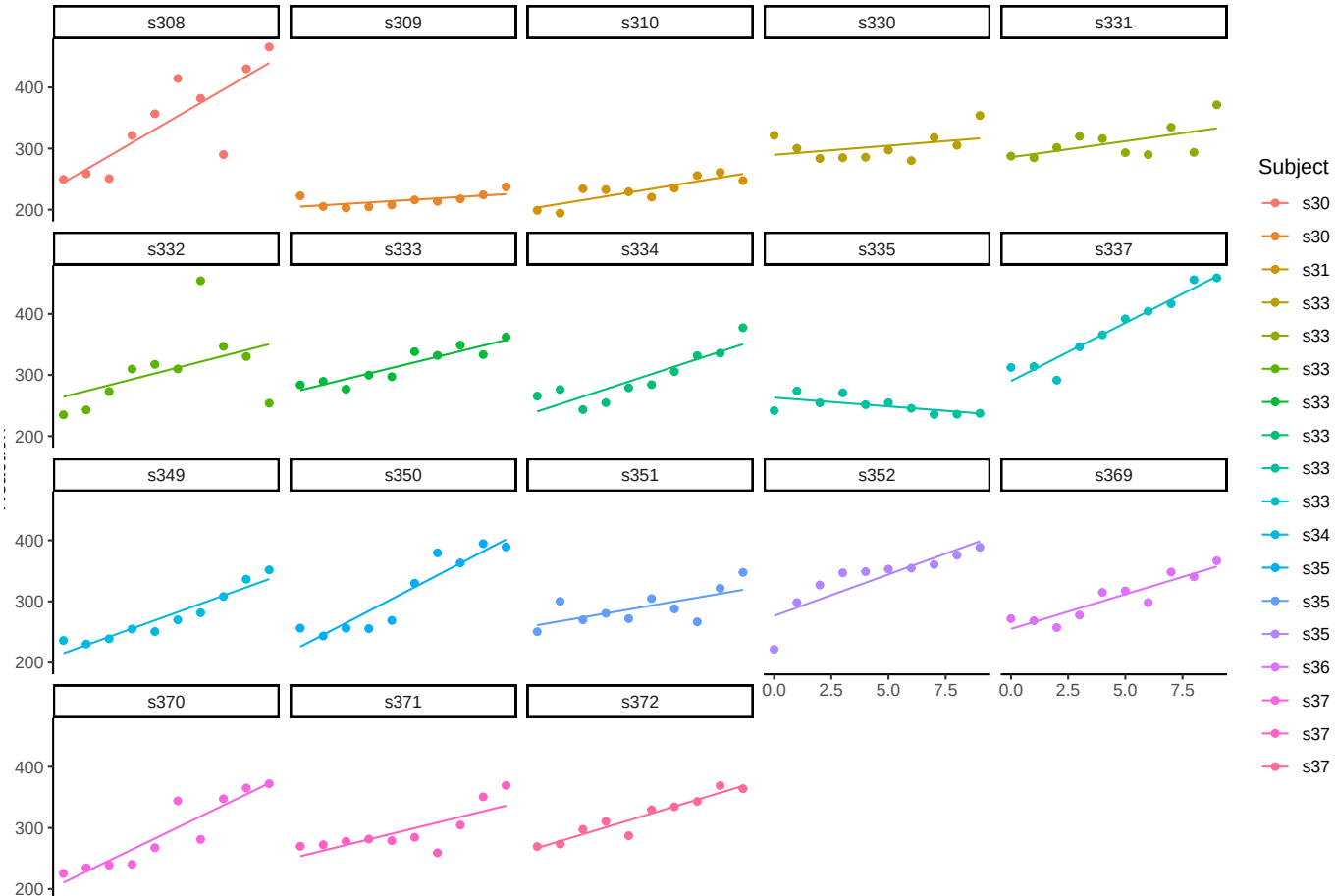
Generalized linear models

Binary logistic regression

```
> M_glm <- glm(survived ~ sex,
+              data = titanic_df,
+              family=binomial)
>
> M_brm <- brm(survived ~ sex,
+              data = titanic_df,
+              cores = 2,
+              family = bernoulli(),
+              prior = set_prior('normal(0, 100)'),
+              save_all_pars = T
+ )
```

Multilevel linear models

```
> ggplot(sleep_df,
+        aes(x=Days, y=Reaction, col=Subject)
+ ) + geom_point() +
+   stat_smooth(method='lm', se=F, size=0.5) +
+   facet_wrap(~Subject) +
+   theme_classic()
```



Random intercepts

```
> M_0_lmer <- lmer(Reaction ~ Days + (1|Subject),
+                  data = sleep_df)
>
> M_0_brm <- brm(Reaction ~ Days + (1|Subject),
+                cores = 2,
+                prior = set_prior('normal(0, 100)'), # flat(ish
+                save_all_pars = T,
+                data = sleep_df)
```

Random intercepts and random slopes model

```
> M_1_lmer <- lmer(Reaction ~ Days + (Days|Subject),
+                  data = sleep_df)
>
> M_1_brm <- brm(Reaction ~ Days + (Days|Subject),
+                cores = 2,
+                prior = set_prior('normal(0, 100)'),
+                save_all_pars = T,
+                data = sleep_df)
```

Nested multilevel linear models

```
> M_2_brm <- brm(like ~ sex + PrivPub + (1|school) + (1|Class),
+                cores = 2,
+                prior = set_prior('normal(0, 100)'),
+                save_all_pars = T,
+                data = science_df)
```

Ordinal logistic

```
> M_3_brm <- brm(like ~ sex + PrivPub + (1|school) + (1|Class),
+                cores = 2,
+                prior = set_prior('normal(0, 100)'),
+                save_all_pars = T,
+                family=cumulative("logit"),
+                data = science_df)
```

Multilevel logistic regression

```
> sleep_df %<>% mutate(fast_rt = Reaction < median(Reaction))
>
> # consider using control = list(adapt_delta = 0.95)
> M_4_brm <- brm(fast_rt ~ Days + (Days|Subject),
+                family = bernoulli(),
+                cores = 2,
+                prior = set_prior('normal(0, 100)'),
+                save_all_pars = T,
+                data = sleep_df)
```