Bayesian Statistics with Stan

Packages for this section

```r
library(tidyverse)
library(rstan)
```

Bayesian and frequentist inference

- The inference philosophy that we have learned so far says that:
  - parameters to be estimated are *fixed* but *unknown*
  - Data random; if we took another sample we'd get different data.
- This is called "frequentist" or "repeated-sampling" inference.
- Bayesian inference says:
  - *parameters* are random, *data* is *given*
- Ingredients:
  - **prior distribution**: distribution of parameters before seeing data.
  - **likelihood**: model for data if the parameters are known
  - **posterior distribution**: distribution of parameters *after* seeing data.

Distribution of parameters

- Instead of having a point or interval estimate of a parameter, we have an entire distribution
- so in Bayesian statistics we can talk about eg.
  - probability that a parameter is bigger than some value
  - probability that a parameter is close to some value
  - probability that one parameter is bigger than another
- Name comes from Bayes' Theorem, which here says *posterior is proportional to likelihood times prior*
- more discussion about this is in **a blog post**.

An example

- Suppose we have these (integer) observations:

```
(x <- c(0, 4, 3, 6, 3, 3, 2, 4))
```

```
## [1] 0 4 3 6 3 3 2 4
```

- Suppose we believe that these come from a Poisson distribution with a mean λ that we want to estimate.
- We need a prior distribution for λ. I will (for some reason) take a $Weibull$ distribution with parameters 1.1 and 6, that has quartiles 2 and 6. Normally this would come from your knowledge of the data-generating *process*.
- The Poisson likelihood can be written down (see over).

Some algebra

- We have $n = 8$ observations x_i, so the Poisson likelihood is proportional to

$$\prod_{i=1}^{n} e^{-\lambda} \lambda^{x_i} = e^{-n\lambda} \lambda^{S},$$

where $S = \sum_{i=1}^{n} x_i$.

- then you write the Weibull prior density (as a function of λ):

$$C(\lambda/6)^{0.1} e^{-(\lambda/6)^{1.1}}$$

where C is a constant.

- and then you multiply these together and try to recognize the distributional form. Only, here you can't. The powers 0.1 and 1.1 get in the way.

Sampling from the posterior distribution

- Wouldn't it be nice if we could just *sample* from the posterior distribution? Then we would be able to compute it as accurately as we want.
- Metropolis and Hastings: devise a Markov chain (C62) whose limiting distribution is the posterior you want, and then sample from that Markov chain (easy), allowing enough time to get close enough to the limiting distribution.
- Stan: uses a modern variant that is more efficient (called Hamiltonian Monte Carlo), implemented in R package `rstan`.
- Write Stan code in a file, compile it and sample from it.

Components of Stan code: the model

```
model {
// likelihood
x ~ poisson(lambda);
}
```

This is how you say "X has a Poisson distribution with mean λ". **Note that lines of Stan code have semicolons on the end.**

Components of Stan code: the prior distribution

```
model {
// prior
lambda ~ weibull(1.1, 6);
// likelihood
x ~ poisson(lambda);
}
```

Components of Stan code: data and parameters (first in the Stan code)

```
data {
int x[8];
}

parameters {
real<lower=0> lambda;
}
```

Compile and sample from the model

```r
poisson1_code <- stan_model(file = "poisson1.stan")
```

- set up data

```r
poisson1_data <- list(x = x)
```

- sample

```r
poisson1_fit <- sampling(poisson1_code, data = poisson1_data)
```

The output

```
poisson1_fit
```

```
## Inference for Stan model: poisson1.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4
##
##        mean se_mean   sd 2.5%  25%  50%  75% 97.5%
## lambda 3.18    0.02 0.63 2.06 2.74 3.13 3.57  4.51
## lp__   3.74    0.02 0.74 1.61 3.57 4.02 4.20  4.26
##        n_eff Rhat
## lambda  1381    1
## lp__    1723    1
##
## Samples were drawn using NUTS(diag_e) at Mon Dec  7 14:30:4
## For each parameter, n_eff is a crude measure of effective s
## and Rhat is the potential scale reduction factor on split c
## convergence, Rhat=1).
```

Comments

- This summarizes the posterior distribution of λ
- the posterior mean is 3.20
- with a 95% posterior interval of 2.10 to 4.56.
- The probability that λ is between these two values really is 95%.

Making the code more general

- The coder in you is probably offended by hard-coding the sample size and the parameters of the prior distribution. More generally:

```
data {
  int<lower=1> n;
  real<lower=0> a;
  real<lower=0> b;
  int x[n];
}
...
model {
// prior
lambda ~ weibull(a, b);
// likelihood
x ~ poisson(lambda);
}
```

Set up again and sample:

- Compile again:

```r
poisson2_code <- stan_model(file = "poisson2.stan")
```

- set up the data again including the new things we need:

```r
poisson2_data <- list(x = x, n = length(x), a = 1.1, b = 6)
```

- sample again

```r
poisson2_fit <- sampling(poisson1_code, data = poisson2_data)
```

output should be the same (to within randomness)

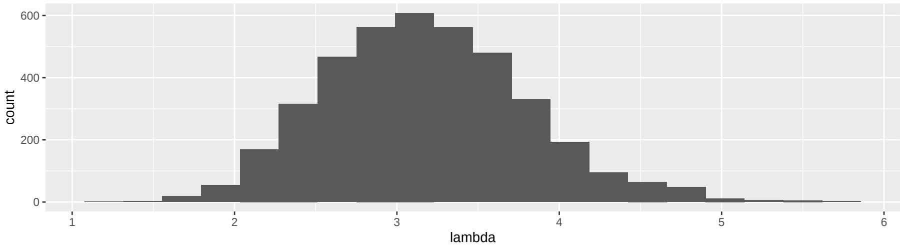
```
poisson2_fit
```

```
## Inference for Stan model: poisson1.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4
##
##        mean se_mean   sd 2.5%  25%  50%  75% 97.5%
## lambda 3.18    0.02 0.62 2.07 2.74 3.15 3.58  4.55
## lp__   3.75    0.02 0.72 1.77 3.60 4.03 4.21  4.26
##        n_eff Rhat
## lambda  1550    1
## lp__    1609    1
##
## Samples were drawn using NUTS(diag_e) at Mon Dec  7 14:30:4
## For each parameter, n_eff is a crude measure of effective s
## and Rhat is the potential scale reduction factor on split c
## convergence, Rhat=1).
```

Extracting actual sampled values

- rstan has extract for this. There is also an extract in dplyr: make sure you have the right one.

```
poisson2_out <- extract(poisson2_fit)
ggplot(tibble(lambda = poisson2_out$lambda), aes(x = lambda))
  geom_histogram(bins = 20)
```



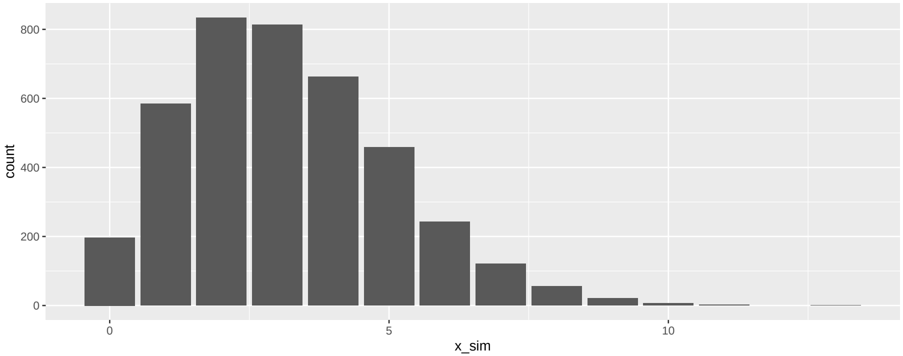
Posterior predictive distribution

- Another use for the actual sampled values is to see what kind of *response* values we might get in the future. This should look something like our data. For a Poisson distribution, the response values are integers:

```
tibble(lambda = poisson2_out$lambda) %>%
  mutate(x_sim = map_int(lambda, ~ rpois(1, .))) -> d
```

A bar chart:

```r
ggplot(d, aes(x = x_sim)) + geom_bar()
```



Comparison

Our actual data values were these:

```
x
```

```
## [1] 0 4 3 6 3 3 2 4
```

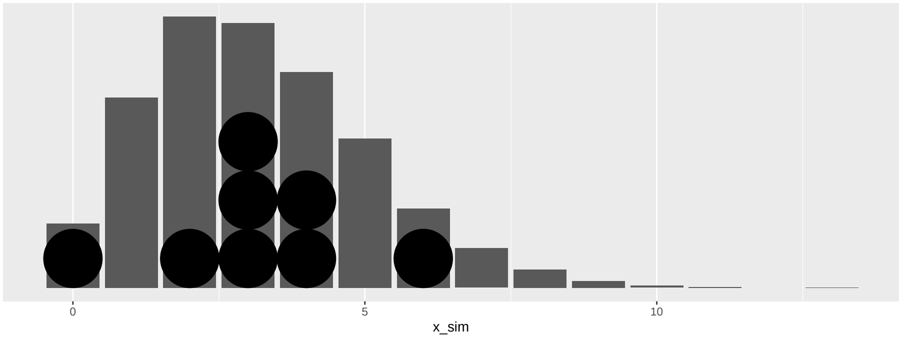
- None of these are very unlikely according to our posterior predictive distribution, so our model is believable.
- Or make a plot: a bar chart with the data on it as well (over):

```
ggplot(d, aes(x = x_sim)) + geom_bar() +
  geom_dotplot(data = tibble(x), aes(x = x), binwidth = 1) +
  scale_y_continuous(NULL, breaks = NULL) -> g
```

- This also shows that the distribution of the data conforms well enough to the posterior predictive distribution (over).

The plot

```
g
```



Analysis of variance, the Bayesian way

Recall the jumping rats data:

```
my_url <- "http://www.utsc.utoronto.ca/~butler/c32/jumping.txt
rats0 <- read_delim(my_url, " ")
rats0 %>% sample_n(6) # random sample of rows
```

group	density
Control	600
Highjump	626
Lowjump	596
Lowjump	588
Highjump	643
Lowjump	594

Our aims here

- Estimate the mean bone density of all rats under each of the experimental conditions
- Model: given the group means, each observation normally distributed with common variance σ^2
- Three parameters to estimate, plus the common variance.
- Obtain posterior distributions for the group means.
- Ask whether the posterior distributions of these means are sufficiently different.

Numbering the groups

- Stan doesn't handle categorical variables (everything is `real` or `int`).
- Turn the groups into group *numbers* first.
- Take opportunity to put groups in logical order:

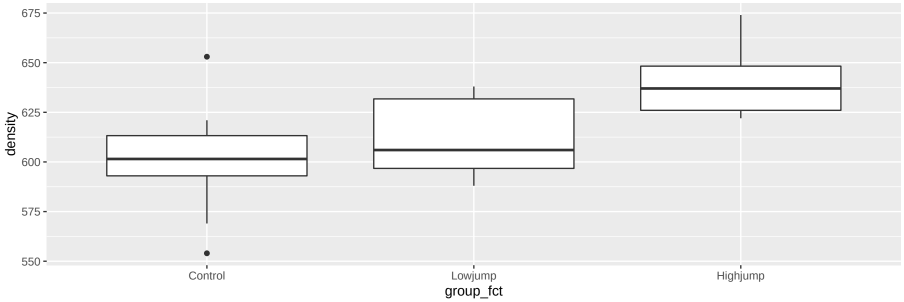
```
rats0 %>% mutate(
  group_fct = fct_inorder(group),
  group_no = as.integer(group_fct)
) -> rats
rats %>% sample_n(4)
```

group	density	group_fct	group_no
Highjump	622	Highjump	3
Lowjump	635	Lowjump	2
Lowjump	605	Lowjump	2
Highjump	622	Highjump	3

Plotting the data 1/2

Most obviously, boxplots:

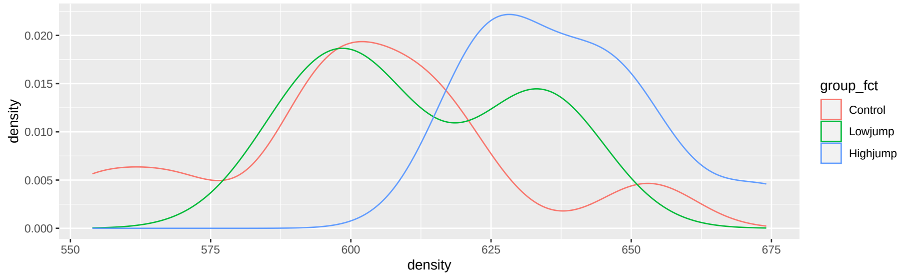
```
ggplot(rats, aes(x = group_fct, y = density)) + geom_boxplot()
```



Plotting the data 2/2

Another way: density plot (smoothed out histogram); can distinguish groups by colours:

```r
ggplot(rats, aes(x = density, colour = group_fct)) +
  geom_density()
```



The procedure

- For each observation, find out which (numeric) group it belongs to,
- then model it as having a normal distribution with that group's mean and the common variance.
- Stan does `for` loops.

The model part

Suppose we have `n_obs` observations:

```
model {
  // likelihood
  for (i in 1:n_obs) {
    g=group_no[i];
    density[i] ~ normal(mu[g], sigma);
  }
}
```

The variables here

- `n_obs` is data.
- g is a temporary integer variable only used here
- `i` is only used in the loop (integer) and does not need to be declared
- `density` is data, a real vector of length `n_obs`
- `mu` is a parameter, a real vector of length 3 (3 groups)
- `sigma` is a real parameter

`mu` and `sigma` need prior distributions:

- for `mu`, each component independently normal with mean 600 and SD 50 (my guess at how big and variable they will be)
- for `sigma`, chi-squared with 50 df (my guess at typical amount of variability from obs to obs)

Complete the `model` section:

```
model {
  int g;
  // priors
  mu ~ normal(600, 50);
  sigma ~ chi_square(50);
  // likelihood
  for (i in 1:n_obs) {
    g=group_no[i];
    density[i] ~ normal(mu[g], sigma);
  }
}
```

Parameters

The elements of `mu`, one per group, and also `sigma`, scalar:

```
parameters {
  real mu[n_group];
  real<lower=0> sigma;
}
```

- `sigma` has to be positive. Declare it so here, so that the sampling runs smoothly.
- declare `n_group` in data section

Data

Everything else:

```
data {
  int n_obs;
  int n_group;
  real density[n_obs];
  int<lower=1, upper=n_group> group_no[n_obs];
}
```

Compile

Arrange these in order data, parameters, model in file `anova.stan`, then:

```
anova_compiled <- stan_model("anova.stan")
```

Set up data and sample

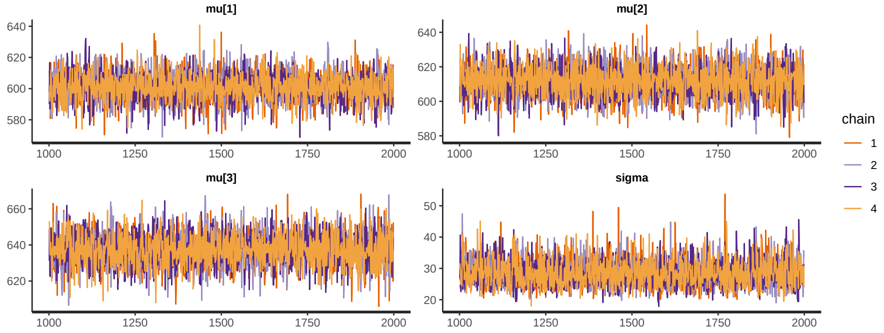
Supply values for *everything* declared in `data`:

```r
anova_data <- list(
  n_obs = 30,
  n_group = 3,
  density = rats$density,
  group_no = rats$group_no
)
anova_samples <- sampling(anova_compiled, data = anova_data)
```

```
##
## SAMPLING FOR MODEL 'anova' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 6e-06 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per trans
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
```

Check that the sampling worked properly

```
traceplot(anova_samples)
```



Comments

- The sampled values for each of the parameters should move freely across their posterior distributions (and not get stuck anywhere).
- This appears to have happened.

Look at the results

```
anova_samples
```

```
## Inference for Stan model: anova.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4
##
##          mean se_mean   sd   2.5%    25%    50%
## mu[1] 600.81    0.14 9.08 582.53 595.06 600.87
## mu[2] 611.82    0.13 8.95 594.26 605.72 611.93
## mu[3] 637.43    0.14 8.88 619.79 631.66 637.61
## sigma  28.59    0.07 4.24  21.54  25.57  28.09
## lp__  -41.02    0.04 1.48 -44.83 -41.77 -40.67
##          75%  97.5% n_eff Rhat
## mu[1] 606.72 618.88  4331    1
## mu[2] 617.74 629.38  4711    1
## mu[3] 643.35 654.68  3769    1
## sigma  31.15  38.22  3274    1
```

Comments

- The posterior 95% intervals for control (group 1) and highjump (group 3) do not quite overlap, suggesting that these exercise groups really are different.
- Bayesian approach does not normally do tests: look at posterior distributions and decide whether they are different enough to be worth treating as different.

Plotting the posterior distributions for the mu

- Extract the sampled mu values (matrix):

```r
anova_ext <- extract(anova_samples)
head(anova_ext$mu)
```

```
##
## iterations      [,1]     [,2]     [,3]
##       [1,] 621.6267 603.2633 642.0971
##       [2,] 581.5891 599.0321 643.1573
##       [3,] 600.7369 615.3000 625.9011
##       [4,] 600.5661 607.6583 644.9923
##       [5,] 584.3141 609.0487 640.2289
##       [6,] 603.6806 602.5221 633.7430
```

Turn into a data frame, arrange for plotting, name groups

```
cbind(anova_ext$mu, sigma = anova_ext$sigma) %>%
  as_tibble() %>%
  pivot_longer(starts_with("V"), names_to = "group", values_to
  mutate(group = fct_recode(
    group,
    Control = "V1",
    Lowjump = "V2",
    Highjump = "V3"
  )) -> sims
```

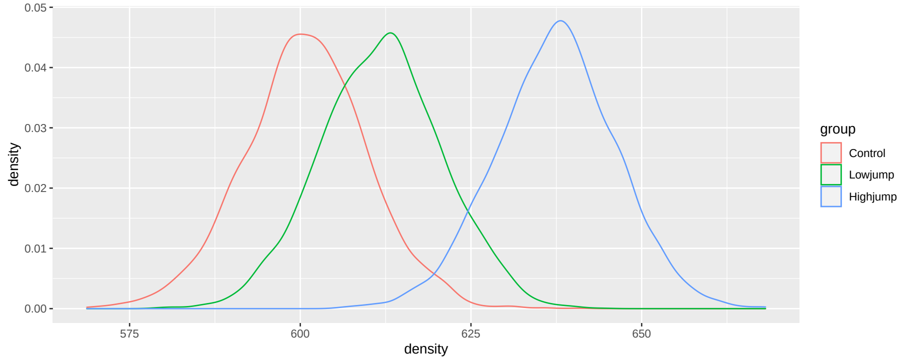
What we have now:

```
sims %>% sample_n(8)
```

sigma	group	density
33.22177	Lowjump	608.3247
25.00378	Highjump	647.3653
30.51957	Lowjump	604.5329
27.80761	Control	605.3842
28.57664	Control	584.4718
27.63376	Highjump	644.8699
25.34673	Lowjump	613.0778
26.58243	Control	615.3927

Density plots of posterior mean distributions

```
ggplot(sims, aes(x = density, colour = group)) + geom_density(
```



Posterior predictive distributions

Randomly sample from posterior means and SDs in `sims`. There are 12000 rows in `sims`:

```
sims %>% mutate(sim_data = rnorm(12000, density, sigma)) -> pp
ppd %>% sample_n(8)
```

sigma	group	density	sim_data
24.84913	Control	596.7946	584.6939
26.91140	Control	610.1608	590.9665
26.36095	Lowjump	612.4005	634.1006
33.54126	Lowjump	615.1805	641.4235
32.25812	Highjump	639.6497	624.2893
26.76857	Highjump	635.1278	644.7062
32.39724	Highjump	647.4838	647.7141
33.89869	Lowjump	608.9650	585.7492

Compare posterior predictive distribution with actual data

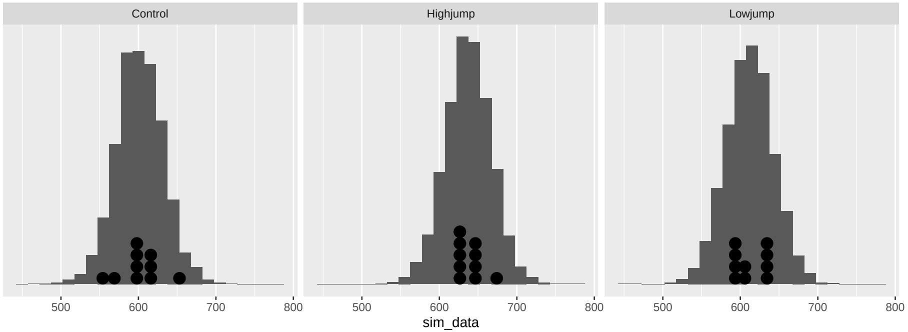
- Check that the model works: distributions of data similar to what we'd predict
- Idea: make plots of posterior predictive distribution, and plot actual data as points on them
- Use facets, one for each treatment group:

```r
my_binwidth <- 15
ggplot(ppd, aes(x = sim_data)) +
  geom_histogram(binwidth = my_binwidth) +
  geom_dotplot(
    data = rats, aes(x = density),
    binwidth = my_binwidth
  ) +
  facet_wrap(~group) +
  scale_y_continuous(NULL, breaks = NULL) -> g
```

- See (over) that the data values are mainly in the middle of the

The plot

```
g
```



Extensions

- if you want a different model other than normal, change distribution in `model` section
- if you want to allow unequal spreads, create `sigma[n_group]` and in model `density[i] ~ normal(mu[g], sigma[g]);`
- Stan will work just fine after you recompile
- very flexible.
- Typical modelling strategy: start simple, add complexity as warranted by data.

Comparing models

- Typically in ANOVA situation, we want to see whether the group means are "really" different.
- We are used to doing an F-test here.
- In regression, have looked at AIC: measure of fit allowing for complexity.
- Bayesian equivalent called `looic`, in package `loo`.

Difficulty:

- Posterior proportional to likelihood times prior:
    - Stan drops proportionality constant
    - but for model comparison, it matters.
- Have to calculate log-likelihood again, not dropping constants this time.

The extra code

```
generated quantities {
  vector[n_obs] log_lik;
  int g;
  for (i in 1:n_obs) {
    g=group_no[i];
    log_lik[i] = normal_lpdf(density[i] | mu[g], sigma);
  }
}
```

Comments

- This is repeat of `model` section, but note no “squiggle”: this explicitly calculates the log of the normal density function for each observation and saves it in an array.
- This section goes at the bottom.

Compile and sample

```
anova_loo_compiled <- stan_model("anova-loo.stan")
```

```
## Trying to compile a simple C file
```

```
anova_data <- list(
  n_obs = 30,
  n_group = 3,
  density = rats$density,
  group_no = rats$group_no
)
anova_loo_samples <- sampling(anova_loo_compiled, data = anova
```

Now we need a null model

- one value of `mu` for all groups
- replace all the `mu[i]` with `mu`
- omit any reference to group numbers (not needed any more)
- leave the `data` section as is (so can use previous `anova_data`).

The null Stan code 1/2

```
data {
  int n_obs;
  int n_group;
  real density[n_obs];
  int<lower=1, upper=n_group> group_no[n_obs];
}

parameters {
  real mu;
  real<lower=0> sigma;
}
```

The null Stan code 2/2

```
model {
  // priors
  mu ~ normal(600, 50);
  sigma ~ chi_square(50);
  // likelihood
  for (i in 1:n_obs) {
    density[i] ~ normal(mu, sigma);
  }
}

generated quantities {
  vector[n_obs] log_lik;
  for (i in 1:n_obs) {
    log_lik[i] = normal_lpdf(density[i] | mu, sigma);
  }
}
```

Compile and sample again

```r
anova_loo_null_compiled <- stan_model("anova_loo_null.stan")
```

```
## Trying to compile a simple C file
```

```r
anova_loo_null_samples <- sampling(anova_loo_null_compiled, da
```

Compare the fits of the two models

Setup

```r
library(loo)
log_lik_a <- extract_log_lik(anova_loo_samples,
  merge_chains = F
)
log_lik_0 <- extract_log_lik(anova_loo_null_samples,
  merge_chains = F
)
r_eff_a <- relative_eff(log_lik_a)
r_eff_0 <- relative_eff(log_lik_0)
```

Results 1/2

```r
loo(log_lik_a, r_eff = r_eff_a)
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -139.0 2.6
## p_loo         2.4 0.6
## looic       278.0 5.2
## ------
## 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.
```

Results 2/2

```r
loo(log_lik_0, r_eff = r_eff_0)
```

```
##
## Computed from 4000 by 30 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo   -142.8 2.8
## p_loo         1.3 0.4
## looic       285.7 5.6
## ------
## 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.
```

Comments

- Look at `looic`: smaller value is better, allowing for complexity of model
- For separate means, 277.7; for one single mean, 285.7.
- Prefer the model with separate means, one for each group.