

# Bayesian Statistics with Stan

## Packages for this section

Installation instructions for the last three of these are below.

```
library(tidyverse)
library(cmdstanr)
library(posterior)
library(bayesplot)
```

# Installation 1/2

- cmdstanr:

```
install.packages("cmdstanr",
                 repos = c("https://stan-dev.r-universe.dev",
                           "https://cloud.r-project.org"))
```

- posterior and bayesplot, from the same place:

```
install.packages("posterior",
                 repos = c("https://stan-dev.r-universe.dev",
                           "https://cloud.r-project.org"))
install.packages("bayesplot",
                 repos = c("https://stan-dev.r-universe.dev",
                           "https://cloud.r-project.org"))
```

## Installation 2/2

Then, to check that you have the C++ stuff needed to compile Stan code:

```
check_cmdstan_toolchain()
```

which should produce output like The C++ toolchain required for CmdStan is setup properly!, and then:

```
install_cmdstan(cores = 6)
```

If you happen to know how many cores (processors) your computer has, insert the appropriate number. (My new laptop has 8 and my desktop 6.)

All of this is done once. If you have problems, go [here \(link\)](#).

## Bayesian and frequentist inference 1/2

- 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 and frequentist inference 2/2

- 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  $\lambda$  that we want to estimate.
- We need a prior distribution for  $\lambda$ . 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  $\lambda$ ):

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

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 packages `cmdstanr`.
- 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  $\lambda$ ”. **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 {  
    array[8] int x;  
}
```

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

## Compile and sample from the model 1/2

- compile

```
poisson1 <- cmdstan_model("poisson1.stan")
```

```
poisson1
```

```
// Estimating Poisson mean
```

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

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

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

## Compile and sample from the model 2/2

- set up data

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

```
$x
[1] 0 4 3 6 3 3 2 4
```

- sample (output is (very) long)

```
poisson1_fit <- poisson1$sample(data = poisson1_data)
```

Running MCMC with 4 sequential chains...

```
Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 1 Iteration:   100 / 2000 [  5%]  (Warmup)
Chain 1 Iteration:   200 / 2000 [ 10%]  (Warmup)
Chain 1 Iteration:   300 / 2000 [ 15%]  (Warmup)
Chain 1 Iteration:   400 / 2000 [ 20%]  (Warmup)
Chain 1 Iteration:   500 / 2000 [ 25%]  (Warmup)
Chain 1 Iteration:   600 / 2000 [ 30%]  (Warmup)
Chain 1 Iteration:   700 / 2000 [ 35%]  (Warmup)
Chain 1 Iteration:   800 / 2000 [ 40%]  (Warmup)
Chain 1 Iteration:   900 / 2000 [ 45%]  (Warmup)
Chain 1 Iteration:  1000 / 2000 [ 50%]  (Warmup)
Chain 1 Iteration:  1100 / 2000 [ 55%]  (Warmup)
Chain 1 Iteration:  1200 / 2000 [ 60%]  (Warmup)
Chain 1 Iteration:  1300 / 2000 [ 65%]  (Warmup)
Chain 1 Iteration:  1400 / 2000 [ 70%]  (Warmup)
Chain 1 Iteration:  1500 / 2000 [ 75%]  (Warmup)
Chain 1 Iteration:  1600 / 2000 [ 80%]  (Warmup)
Chain 1 Iteration:  1700 / 2000 [ 85%]  (Warmup)
Chain 1 Iteration:  1800 / 2000 [ 90%]  (Warmup)
Chain 1 Iteration:  1900 / 2000 [ 95%]  (Warmup)
Chain 1 Iteration:  2000 / 2000 [100%]  (Warmup)
```

## The output

```
poisson1_fit
```

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
lp__	3.75	4.04	0.73	0.30	2.33	4.26	1.00	1844	2236
lambda	3.22	3.18	0.63	0.62	2.26	4.33	1.00	1664	1955

## Comments

- This summarizes the posterior distribution of  $\lambda$
- the posterior mean is 3.22
- with a 90% posterior interval of 2.26 to 4.33.
- The probability that  $\lambda$  is between these two values really is 90%.

## 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;  
    array[n] int x;  
}  
...  
model {  
    // prior  
    lambda ~ weibull(a, b);  
    // likelihood  
    x ~ poisson(lambda);  
}
```

## Set up again and sample:

- Compile again:

```
poisson2 <- cmdstan_model("poisson2.stan")
```

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

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

```
$x
[1] 0 4 3 6 3 3 2 4
```

```
$n
[1] 8
```

```
$a
[1] 1.1
```

```
$b
[1] 6
```

## Sample again

Output should be the same (to within randomness):

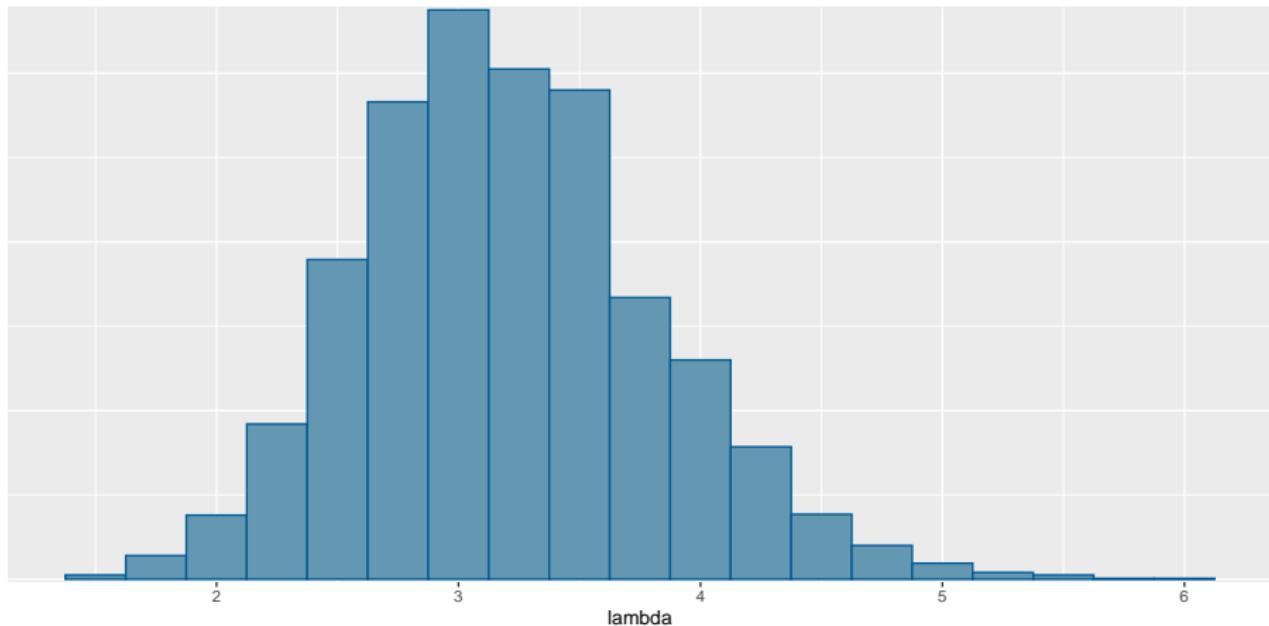
```
poisson2_fit <- poisson2$sample(data = poisson2_data)
```

```
poisson2_fit
```

variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk	ess_tail
lp__	3.77	4.05	0.71	0.28	2.38	4.25	1.00	1835	1943
lambda	3.20	3.17	0.61	0.59	2.27	4.27	1.00	1448	1612

## Picture of posterior

```
mcmc_hist(poisson2_fit$draws("lambda"), binwidth = 0.25)
```



## Extracting actual sampled values

A little awkward at first:

```
str(poission2_fit$draws())
```

```
'draws_array' num [1:1000, 1:4, 1:2] 4.16 3.97 4.25 3.73 2.71
- attr(*, "dimnames")=List of 3
..$ iteration: chr [1:1000] "1" "2" "3" "4" ...
..$ chain     : chr [1:4] "1" "2" "3" "4"
..$ variable  : chr [1:2] "lp__" "lambda"
```

A 3-dimensional array. A dataframe would be much better.

## Sampled values as dataframe

```
as_draws_df(poisson2_fit$draws()) %>%
  as_tibble() -> poisson2_draws
poisson2_draws
```

```
# A tibble: 4,000 x 5
  lp__ lambda .chain .iteration .draw
  <dbl>  <dbl>   <int>     <int> <int>
1  4.16   3.48     1         1      1
2  3.97   2.74     1         2      2
3  4.25   3.26     1         3      3
4  3.73   3.88     1         4      4
5  2.71   4.42     1         5      5
6  2.71   4.42     1         6      6
7  1.13   5.02     1         7      7
8  3.92   2.71     1         8      8
9  4.14   2.90     1         9      9
10 4.26   3.20     1        10     10
```

## 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:

```
poisson2_draws %>%
  rowwise() %>%
  mutate(xsim = rpois(1, lambda)) -> d
```

## The simulated posterior distribution (in xsim)

```
d %>% select(lambda, xsim)
```

```
# A tibble: 4,000 x 2
```

```
# Rowwise:
```

```
lambda  xsim
```

```
<dbl> <int>
```

```
1 3.48     4
```

```
2 2.74     1
```

```
3 3.26     3
```

```
4 3.88     3
```

```
5 4.42     4
```

```
6 4.42     4
```

```
7 5.02     7
```

```
8 2.71     5
```

```
9 2.90     4
```

```
10 3.20    2
```

```
# i 3,990 more rows
```

## Comparison

Our actual data values were these:

```
x
```

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

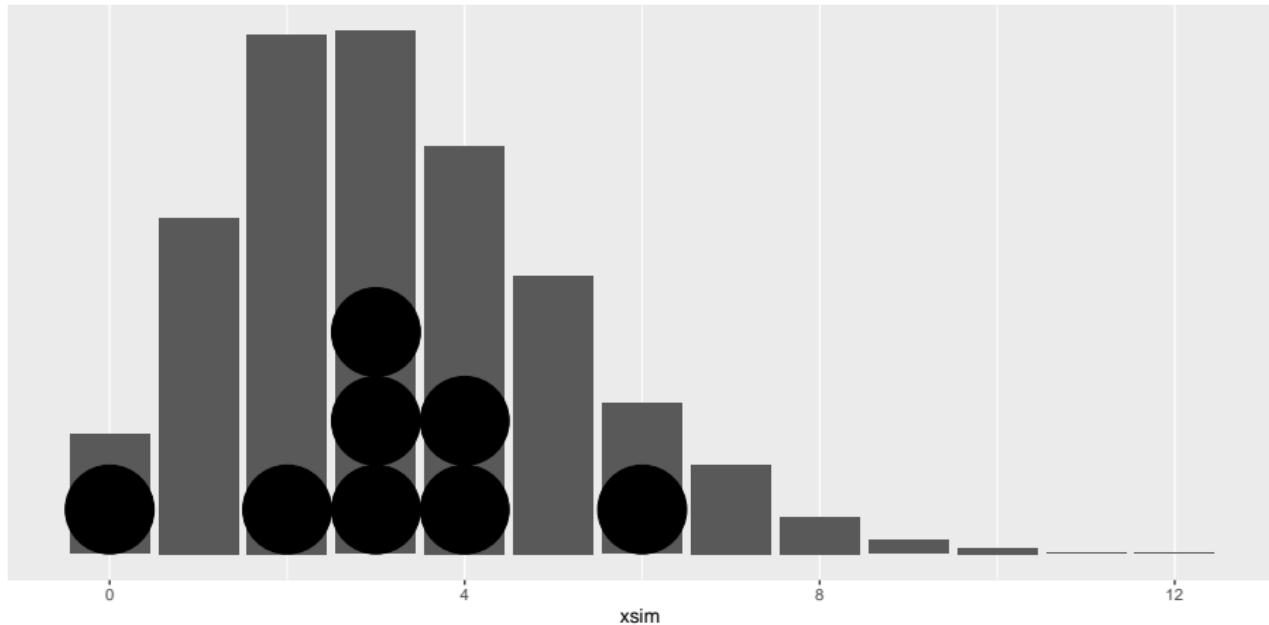
- None of these are very unlikely according to our posterior predictive distribution, so our model is believable.
- Or make a plot: a bar chart with the data on it as well (over):

```
ggplot(d, aes(x = xsim)) + 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



## Do they have the same distribution?

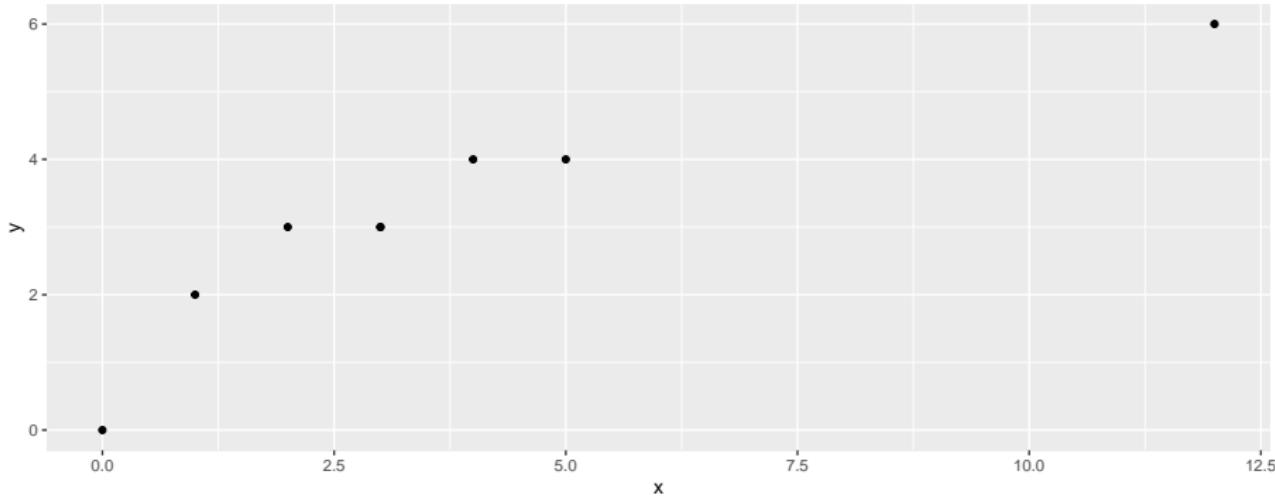
```
qqplot(d$xsim, x, plot.it = FALSE) %>% as_tibble() -> dd  
dd
```

```
# A tibble: 8 x 2
```

	x	y
	<dbl>	<dbl>
1	0	0
2	1	2
3	2	3
4	3	3
5	3	3
6	4	4
7	5	4
8	12	6

## The plot

```
ggplot(dd, aes(x=x, y=y)) + geom_point()
```



the observed zero is a bit too small compared to expected (from the posterior), but the other points seem pretty well on a line.

# Analysis of variance, the Bayesian way

Recall the jumping rats data:

```
my_url <-
  "http://ritsokiguess.site/datafiles/jumping.txt"
rats0 <- read_delim(my_url, " ")
rats0
```

```
# A tibble: 30 x 2
  group    density
  <chr>     <dbl>
1 Control    611
2 Control    621
3 Control    614
4 Control    593
5 Control    593
6 Control    653
7 Control    600
8 Control    554
9 Control    603
10 Control   569
```

## 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  $\sigma^2$
- 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 1/2

- 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
```

## Numbering the groups 2/2

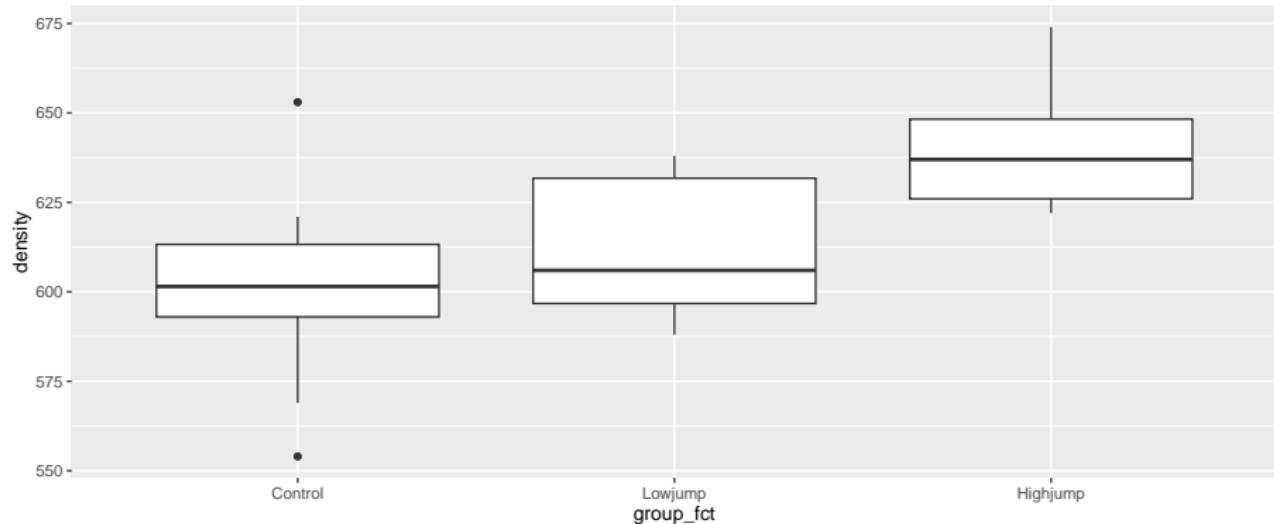
rats

```
# A tibble: 30 x 4
  group    density group_fct group_no
  <chr>     <dbl>  <fct>      <int>
1 Control     611 Control       1
2 Control     621 Control       1
3 Control     614 Control       1
4 Control     593 Control       1
5 Control     593 Control       1
6 Control     653 Control       1
7 Control     600 Control       1
8 Control     554 Control       1
9 Control     603 Control       1
10 Control    569 Control       1
# i 20 more rows
```

## Plotting the data 1/2

Most obviously, boxplots:

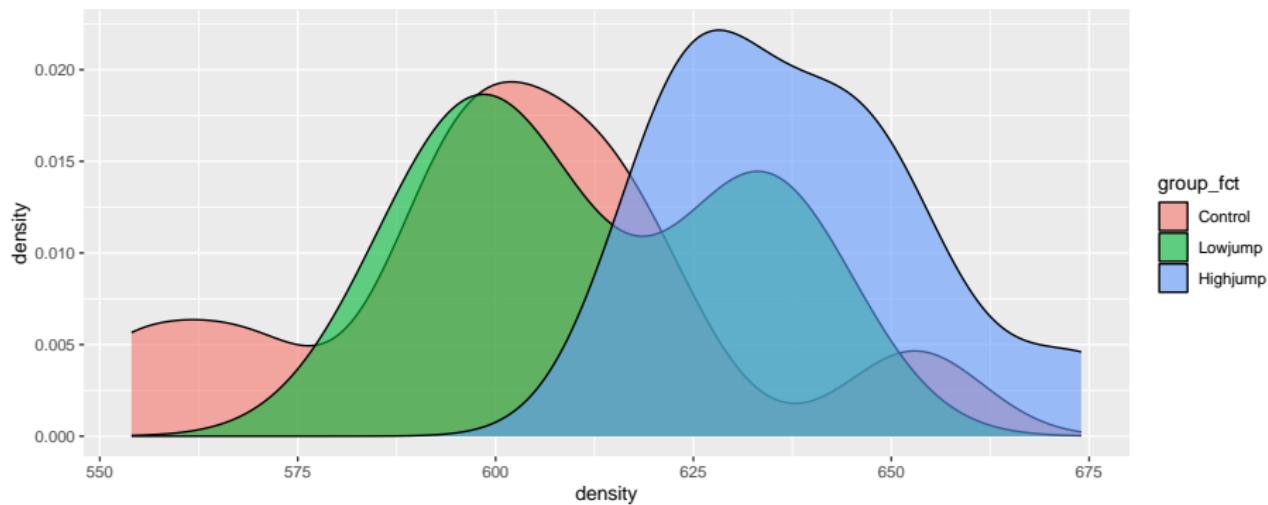
```
ggplot(rats, aes(x = group_fct, y = density)) +  
  geom_boxplot()
```



## Plotting the data 2/2

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

```
ggplot(rats, aes(x = density, fill = group_fct)) +  
  geom_density(alpha = 0.6)
```



## 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, lower limit zero:

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

- Declare `sigma` to have lower limit zero here, so that the sampling runs smoothly.
- declare `n_group` in data section

# Data

Everything else:

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

## Compile

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

```
anova <- cmdstan_model("anova.stan")
```

## Set up data and sample

Supply values for *everything* declared in data:

```
anova_data <- list(  
  n_obs = 30,  
  n_group = 3,  
  density = rats$density,  
  group_no = rats$group_no  
)  
anova_fit <- anova$sample(data = anova_data)
```

Running MCMC with 4 sequential chains...

```
Chain 1 Iteration:    1 / 2000 [  0%]  (Warmup)  
Chain 1 Iteration:   100 / 2000 [  5%]  (Warmup)  
Chain 1 Iteration:   200 / 2000 [ 10%]  (Warmup)  
Chain 1 Iteration:   300 / 2000 [ 15%]  (Warmup)  
Chain 1 Iteration:   400 / 2000 [ 20%]  (Warmup)  
Chain 1 Iteration:   500 / 2000 [ 25%]  (Warmup)
```

## Check that the sampling worked properly

```
anova_fit$cmdstan_diagnose()
```

```
Processing csv files: /tmp/RtmpKLIKxv/anova-202511201119-1-351e3f.csv, /tmp/RtmpKLIKxv/202511201119-2-351e3f.csv, /tmp/RtmpKLIKxv/anova-202511201119-3-351e3f.csv, /tmp/RtmpKLIKxv/202511201119-4-351e3f.csv
```

```
Checking sampler transitions treedepth.  
Treedepth satisfactory for all transitions.
```

```
Checking sampler transitions for divergences.  
No divergent transitions found.
```

```
Checking E-BFMI - sampler transitions HMC potential energy.  
E-BFMI satisfactory.
```

```
Effective sample size satisfactory.
```

```
Split R-hat values satisfactory all parameters.
```

```
Processing complete, no problems detected.
```

## Look at the results

```
anova_fit
```

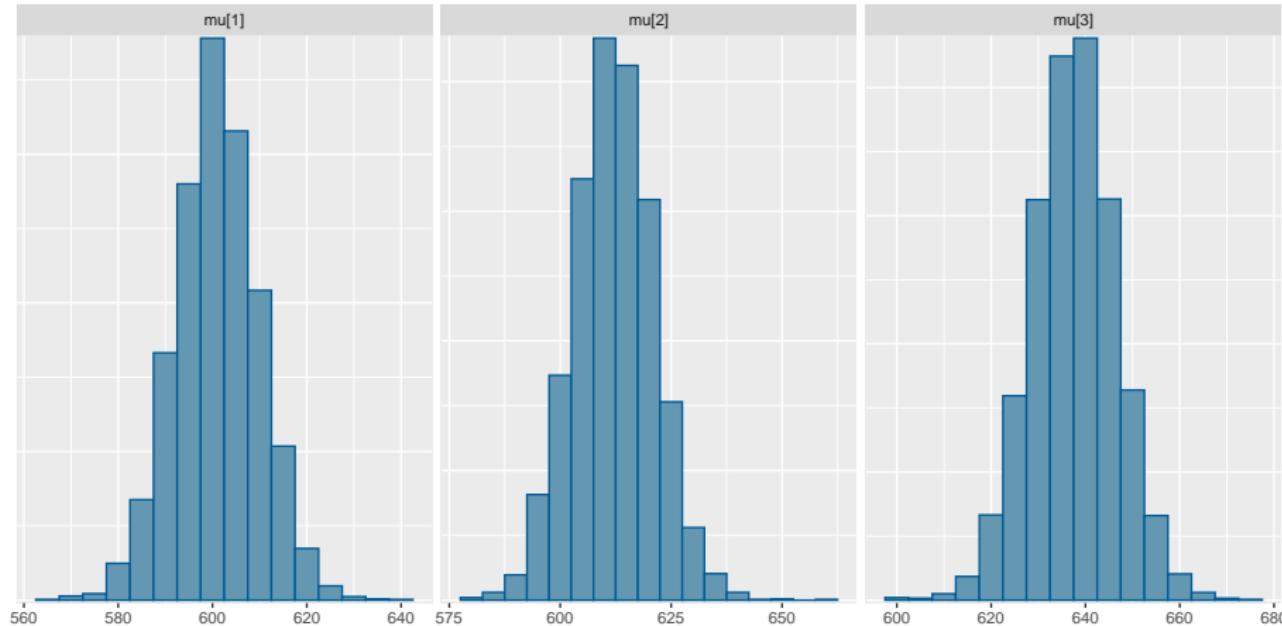
variable	mean	median	sd	mad	q5	q95	rhat	ess_bulk
lp__	-41.02	-40.68	1.51	1.26	-43.87	-39.27	1.00	1867
mu[1]	600.89	600.80	8.92	8.77	586.51	615.65	1.00	4258
mu[2]	612.07	611.99	9.04	8.92	597.10	626.89	1.00	4233
mu[3]	637.55	637.64	9.03	8.76	622.80	651.95	1.00	4516
sigma	28.44	27.97	4.21	4.13	22.34	35.82	1.00	3379

## Comments

- The posterior 90% 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 $\mu$

```
mcmc_hist(anova_fit$draws("mu"), binwidth = 5)
```



## Extract the sampled values

```
as_draws_df(anova_fit$draws()) %>% as_tibble() -> anova_draws  
anova_draws
```

```
# A tibble: 4,000 x 8  
#> # ... with 3,990 more rows  
#> # ... with 8 variables:  
#> #   lp__ <dbl>, `mu[1]` <dbl>, `mu[2]` <dbl>, `mu[3]` <dbl>, sigma <dbl>,  
#> #   .chain <int>, .iteration <int>, .draw <int>  
#> #   ... and 3 more variables:  
#> #   .lp <dbl>, .mu <dbl>, .sigma <dbl>
```

	lp__	`mu[1]`	`mu[2]`	`mu[3]`	sigma	.chain	.iteration	.draw
1	-41.2	611.	628.	637.	29.7	1		1
2	-39.6	606.	620.	640.	28.9	1		2
3	-39.9	603.	618.	643.	30.7	1		3
4	-41.3	593.	595.	636.	30.4	1		4
5	-39.8	603.	620.	631.	25.7	1		5
6	-39.3	599.	618.	637.	24.7	1		6
7	-42.0	614.	611.	636.	36.5	1		7
8	-44.3	586.	621.	643.	20.7	1		8
9	-39.2	601.	616.	634.	28.2	1		9
10	-39.4	598.	611.	644.	27.7	1		10

## Estimated probability that $\mu_3 > \mu_1$

```
anova_draws %>%
  count(`mu[3] > mu[1]) %>%
  mutate(prob = n/sum(n))
```

```
# A tibble: 2 x 3
`mu[3] > mu[1]`      n     prob
<lgl>                  <int>   <dbl>
1 FALSE                 15  0.00375
2 TRUE                 3985 0.996
```

High jumping group almost certainly has larger mean than control group.

## Compare lowjump and control the same way

```
anova_draws %>%
  count(`mu[2] > mu[1]) %>%
  mutate(prob = n/sum(n))
```

```
# A tibble: 2 x 3
`mu[2] > mu[1]`      n   prob
<lgl>                  <int> <dbl>
1 FALSE                751  0.188
2 TRUE                 3249 0.812
```

Likely that lowjump mean higher than control mean, but not a certainty.

## More organizing

- for another plot
  - ▶ make longer
  - ▶ give group values their proper names back

```
anova_draws %>%
  pivot_longer(starts_with("mu"),
               names_to = "group",
               values_to = "bone_density") %>%
  mutate(group = fct_recode(group,
    Control = "mu[1]",
    Lowjump = "mu[2]",
    Highjump = "mu[3]")
)) -> sims
```

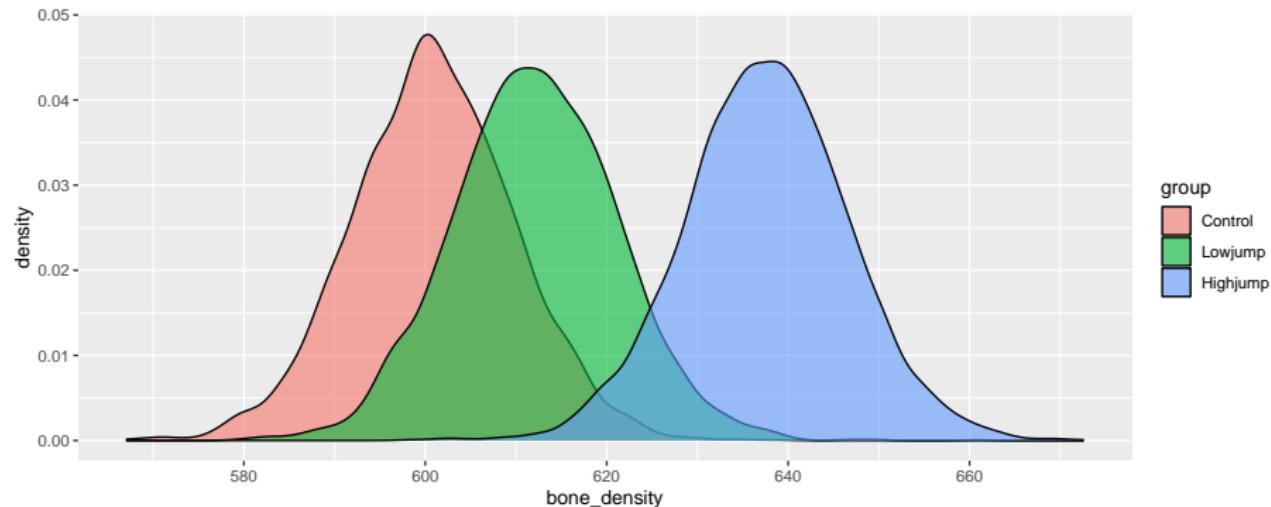
## What we have now:

```
sims
```

```
# A tibble: 12,000 x 7
  lp__ sigma .chain .iteration .draw group bone_density
  <dbl> <dbl> <int>     <int> <int> <fct>      <dbl>
1 -41.2  29.7     1         1     1 Control    611.
2 -41.2  29.7     1         1     1 Lowjump   628.
3 -41.2  29.7     1         1     1 Highjump  637.
4 -39.6  28.9     1         2     2 Control    606.
5 -39.6  28.9     1         2     2 Lowjump   620.
6 -39.6  28.9     1         2     2 Highjump  640.
7 -39.9  30.7     1         3     3 Control    603.
8 -39.9  30.7     1         3     3 Lowjump   618.
9 -39.9  30.7     1         3     3 Highjump  643.
10 -41.3  30.4    1         4     4 Control    593.
# i 11,990 more rows
```

# Density plots of posterior mean distributions

```
ggplot(sims, aes(x = bone_density, fill = group)) +  
  geom_density(alpha = 0.6)
```



## Posterior predictive distributions

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

```
sims %>% mutate(sim_data = rnorm(12000, bone_density,
                                     sigma)) -> ppd
ppd
```

```
# A tibble: 12,000 x 8
   lp__ sigma .chain .iteration .draw group    bone_density sim_data
   <dbl> <dbl> <int>     <int> <int> <fct>      <dbl>       <dbl>
 1 -41.2  29.7     1         1     1 Control     611.       621.
 2 -41.2  29.7     1         1     1 Lowjump    628.       640.
 3 -41.2  29.7     1         1     1 Highjump   637.       618.
 4 -39.6  28.9     1         2     2 Control    606.       580.
 5 -39.6  28.9     1         2     2 Lowjump    620.       623.
 6 -39.6  28.9     1         2     2 Highjump   640.       637.
 7 -39.9  30.7     1         3     3 Control    603.       655.
 8 -39.9  30.7     1         3     3 Lowjump    618.       589.
 9 -39.9  30.7     1         3     3 Highjump   643.       677.
10 -41.3  30.4     1         4     4 Control    593.       598.
# i 11,990 more rows
```

## Compare posterior predictive distribution with actual data

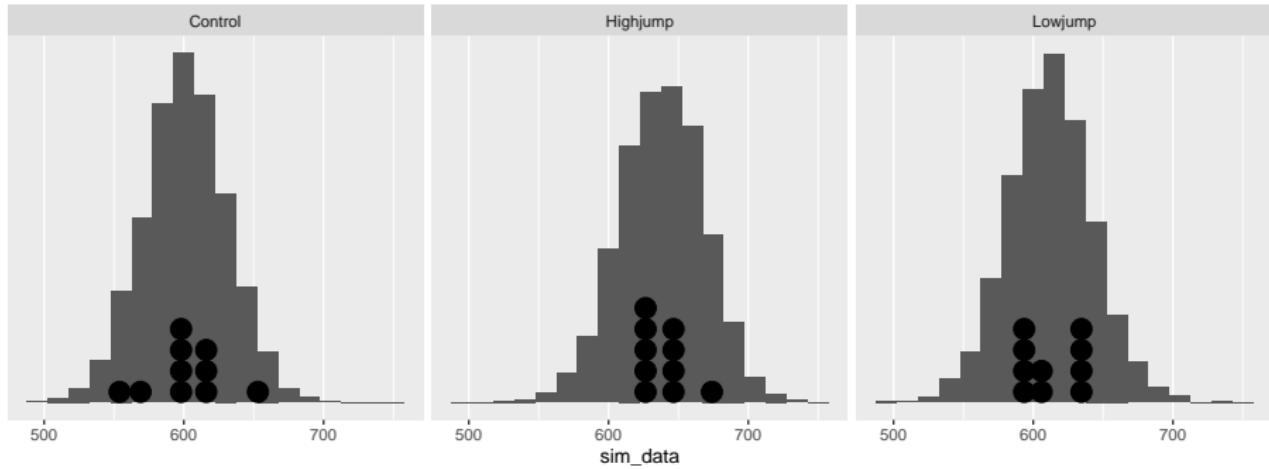
- Check that the model works: distributions of data similar to what we'd predict
- Idea: make plots of posterior predictive distribution, and plot actual data as points on them
- Use facets, one for each treatment group:

```
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 predictive distributions.
- Even for the control group that had outliers.

# 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.