App stat lab exercise 5

```r
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.2     v readr     2.1.4
v forcats   1.0.0     v stringr   1.5.0
v ggplot2   3.4.2     v tibble    3.2.1
v lubridate 1.9.2     v tidyr     1.3.0
v purrr     1.0.1
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to becom
```

```r
library(rstan)
```

```
Loading required package: StanHeaders

rstan version 2.32.5 (Stan version 2.32.2)

For execution on a local, multicore CPU with excess RAM we recommend calling
options(mc.cores = parallel::detectCores()).
To avoid recompilation of unchanged Stan programs, we recommend calling
rstan_options(auto_write = TRUE)
For within-chain threading using `reduce_sum()` or `map_rect()` Stan functions,
change `threads_per_chain` option:
rstan_options(threads_per_chain = 1)


Attaching package: 'rstan'
```

```
The following object is masked from 'package:tidyr':

    extract
```

```r
library(tidybayes)
library(here)
```

```
here() starts at /Users/euijinbaek/STA2201
```

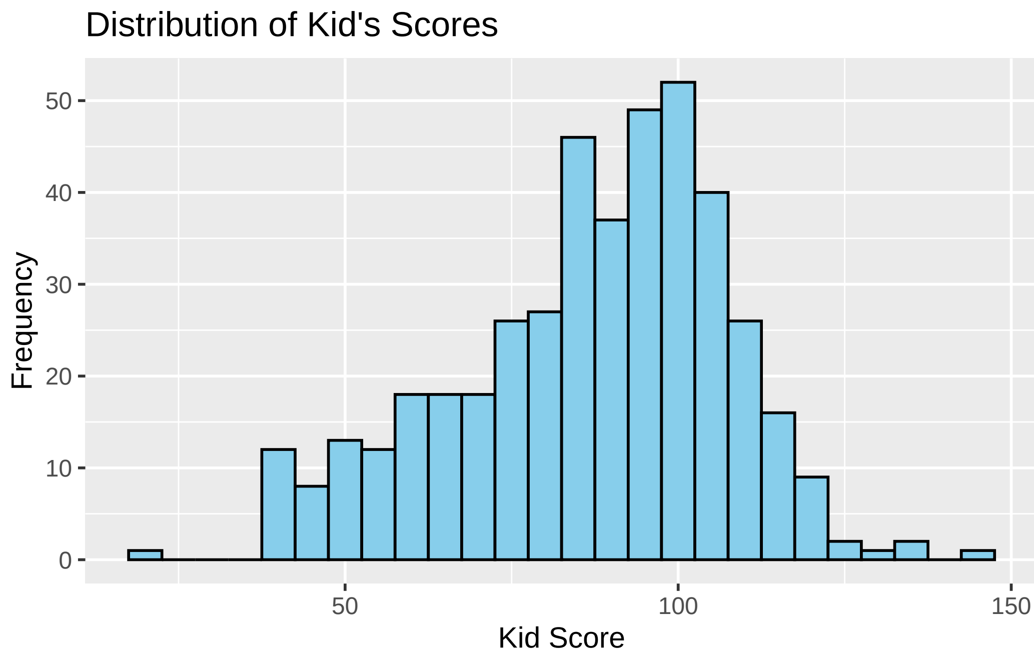
```r
# Data load
kidiq <- read_rds("data/kidiq.RDS")
head(kidiq)
```

```
# A tibble: 6 x 4
  kid_score mom_hs mom_iq mom_age
      <int>  <dbl>  <dbl>   <int>
1        65      1  121.       27
2        98      1   89.4      25
3        85      1  115.       27
4        83      1   99.4      25
5       115      1   92.7      27
6        98      0  108.       18
```

1.

We first use historgram to see distribution of Kid's Scores. Since the distribution is not much different from normal distribution, we could assume that Kid's Scores follow normal distribution. (We assume Normal likelihood)

```r
ggplot(kidiq, aes(x = kid_score)) +
  geom_histogram(binwidth = 5, fill = "skyblue", color = "black") +
  labs(title = "Distribution of Kid's Scores", x = "Kid Score", y = "Frequency")
```



Maybe Mother's eduacation level affect kid's score. Let's see Basic statistics. Mean kid score is higer if Mother's education is ar least high school.

```
kidiq |>
  group_by(mom_hs) |>
  summarize(mean_kid_score = mean(kid_score))
```

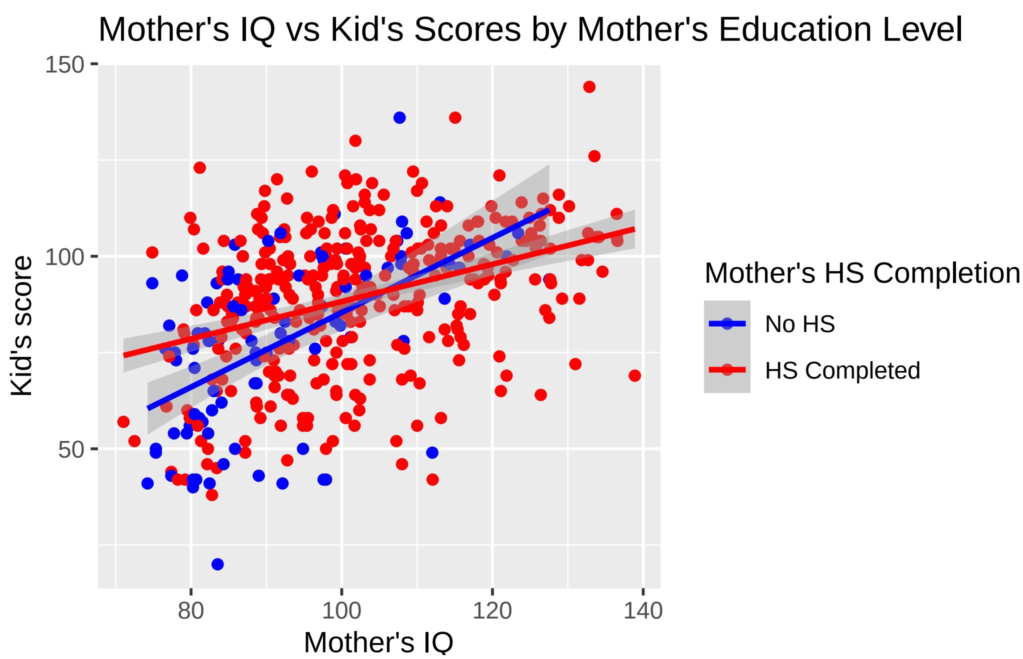
```
# A tibble: 2 x 2
  mom_hs mean_kid_score
   <dbl>          <dbl>
1      0           77.5
2      1           89.3
```

Let's see this relationship in graph. As we can see, there is positive correlation beween kid's score and Mother's IQ overall. The slopes of the two regression lines (one for each group) are positive, which reinforces the observation of a positive correlation. Additionally, the regression line for the mothers who completed high school (red line) is positioned higher than the line for mothers who did not complete high school (blue line), suggesting that completing high school is associated with higher scores for the children, independent of the mother's IQ.

```
kidiq |>
  ggplot(aes(x = mom_iq, y = kid_score, color = as.factor(mom_hs))) +
```

```
geom_point() +
geom_smooth(method = 'lm', aes(group = mom_hs)) +
labs(title = "Mother's IQ vs Kid's Scores by Mother's Education Level",
     y = "Kid's score",
     x = "Mother's IQ",
     color = "Mother's HS Completion") +
scale_color_manual(values = c('0' = 'blue', '1' = 'red'),
                   labels = c('0' = 'No HS', '1' = 'HS Completed'))
```

```
`geom_smooth()` using formula = 'y ~ x'
```



Estimating mean, no covariates

```
y <- kidiq$kid_score
mu0 <- 80
sigma0 <- 10

# named list to input for stan function
data <- list(y = y,
             N = length(y),
             mu0 = mu0,
```

```
              sigma0 = sigma0)
```

Now we can run the model:

```
fit <- stan(file = "code/models/kids2.stan",
            data = data,
            # reducing the iterations a bit to speed things up
            chains = 3,
            iter = 500)
```

```
Warning in readLines(file, warn = TRUE): incomplete final line found on
'/Users/euijinbaek/STA2201/labs/code/models/kids2.stan'
```

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

```
Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
using C compiler: 'Apple clang version 15.0.0 (clang-1500.0.40.1)'
using SDK: 'MacOSX14.0.sdk'
clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Librar
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/St
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/Rc
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/Rc
/Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/
namespace Eigen {
^
/Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/
namespace Eigen {
               ^
               ;
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/St
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/Rc
/Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/
#include <complex>
         ^~~~~~~~~
3 errors generated.
make: *** [foo.o] Error 1

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
```

```
Chain 1: Gradient evaluation took 1.8e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.18 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:   1 / 500 [  0%]  (Warmup)
Chain 1: Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 1: Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 1: Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 1: Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 1: Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 1: Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 1: Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 1: Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 1: Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 1: Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 1: Iteration: 500 / 500 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.004 seconds (Warm-up)
Chain 1:                0.001 seconds (Sampling)
Chain 1:                0.005 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 1e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.01 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:   1 / 500 [  0%]  (Warmup)
Chain 2: Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 2: Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 2: Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 2: Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 2: Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 2: Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 2: Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 2: Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 2: Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 2: Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 2: Iteration: 500 / 500 [100%]  (Sampling)
Chain 2:
```

```
Chain 2:  Elapsed Time: 0.003 seconds (Warm-up)
Chain 2:                0.002 seconds (Sampling)
Chain 2:                0.005 seconds (Total)
Chain 2: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3: 
Chain 3: Gradient evaluation took 2e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.02 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3: 
Chain 3: 
Chain 3: Iteration:   1 / 500 [  0%]  (Warmup)
Chain 3: Iteration:  50 / 500 [ 10%]  (Warmup)
Chain 3: Iteration: 100 / 500 [ 20%]  (Warmup)
Chain 3: Iteration: 150 / 500 [ 30%]  (Warmup)
Chain 3: Iteration: 200 / 500 [ 40%]  (Warmup)
Chain 3: Iteration: 250 / 500 [ 50%]  (Warmup)
Chain 3: Iteration: 251 / 500 [ 50%]  (Sampling)
Chain 3: Iteration: 300 / 500 [ 60%]  (Sampling)
Chain 3: Iteration: 350 / 500 [ 70%]  (Sampling)
Chain 3: Iteration: 400 / 500 [ 80%]  (Sampling)
Chain 3: Iteration: 450 / 500 [ 90%]  (Sampling)
Chain 3: Iteration: 500 / 500 [100%]  (Sampling)
Chain 3: 
Chain 3:  Elapsed Time: 0.004 seconds (Warm-up)
Chain 3:                0.002 seconds (Sampling)
Chain 3:                0.006 seconds (Total)
Chain 3: 

Warning: Bulk Effective Samples Size (ESS) is too low, indicating posterior means and medians
Running the chains for more iterations may help. See
https://mc-stan.org/misc/warnings.html#bulk-ess

Warning: Tail Effective Samples Size (ESS) is too low, indicating posterior variances and tai
Running the chains for more iterations may help. See
https://mc-stan.org/misc/warnings.html#tail-ess
```

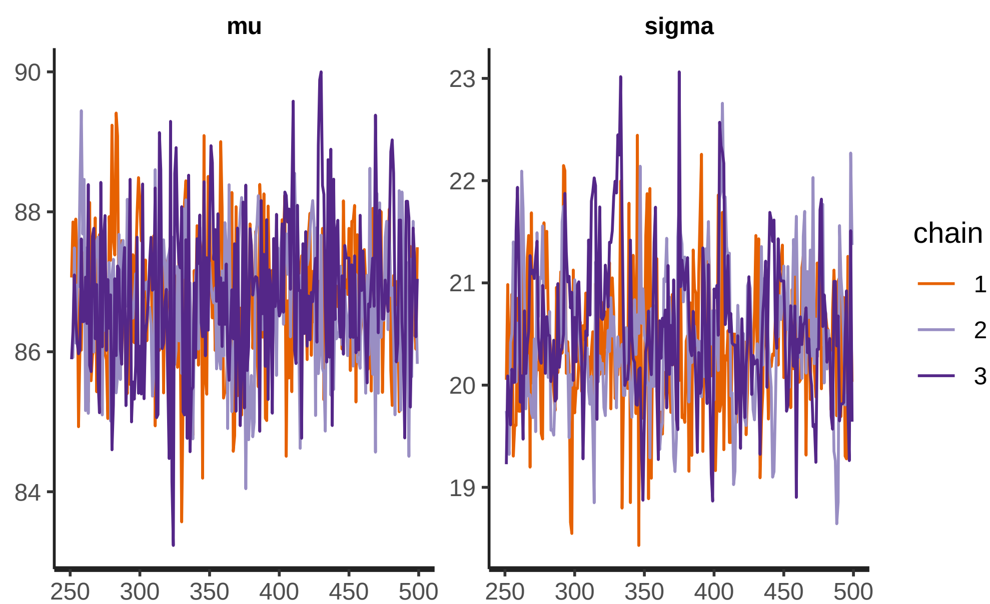
Look at the summary

```
fit
```

```
Inference for Stan model: anon_model.
3 chains, each with iter=500; warmup=250; thin=1;
post-warmup draws per chain=250, total post-warmup draws=750.

          mean se_mean   sd     2.5%      25%      50%      75%    97.5% n_eff
mu       86.78    0.04 1.02    84.80    86.05    86.81    87.49    88.71   653
sigma    20.47    0.04 0.72    19.16    20.01    20.41    20.93    21.99   330
lp__  -1525.85    0.09 1.18 -1529.20 -1526.26 -1525.47 -1525.02 -1524.78   178
      Rhat
mu    1.01
sigma 1.01
lp__  1.02

Samples were drawn using NUTS(diag_e) at Fri Feb 16 23:01:30 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Traceplot

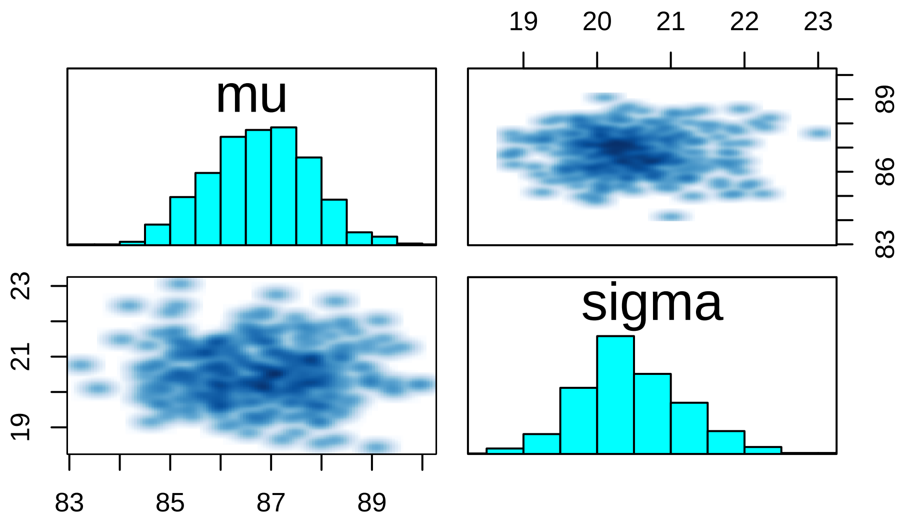
```
traceplot(fit)
```



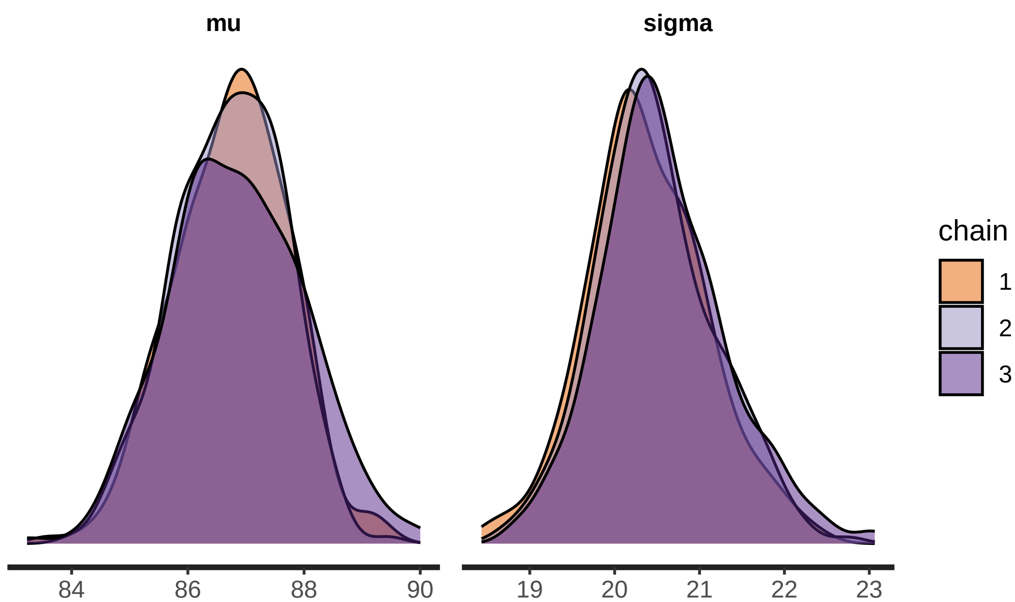
All looks fine.

```
pairs(fit, pars = c("mu", "sigma"))
```

```
Warning in par(usr): argument 1 does not name a graphical parameter

Warning in par(usr): argument 1 does not name a graphical parameter
```




```
stan_dens(fit, separate_chains = TRUE)
```



Understanding output

What does the model actually give us? A number of samples from the posteriors. To see this, we can use `extract` to get the samples.

```
post_samples <- extract(fit)
names(post_samples)
```

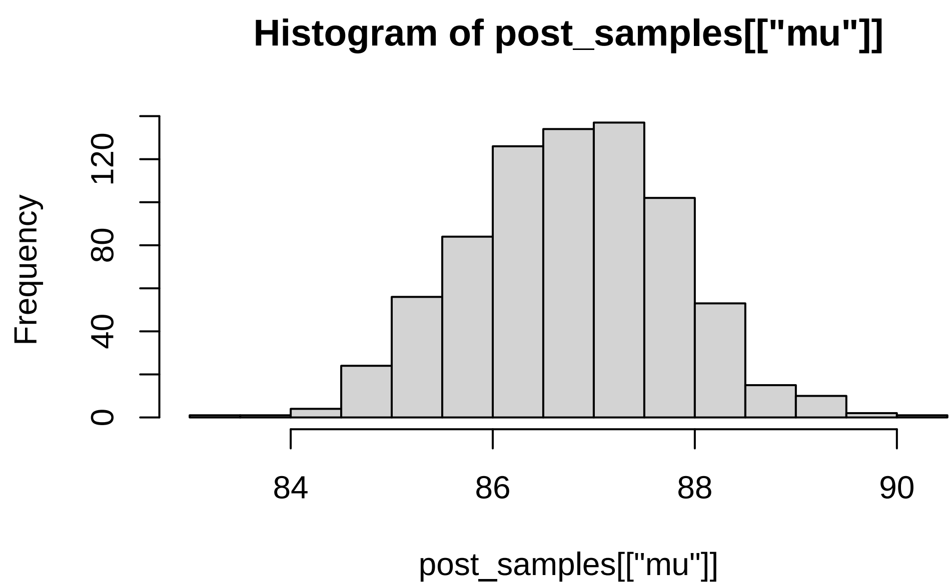
```
[1] "mu"    "sigma" "lp__"
```

```
head(post_samples[["mu"]])
```

```
[1] 86.62505 86.53754 86.68568 86.83702 87.00940 86.95540
```

This is a list, and in this case, each element of the list has 4000 samples. E.g. quickly plot a histogram of mu

```
hist(post_samples[["mu"]])
```




```
median(post_samples[["mu"]])
```

```
[1] 86.80602
```

```r
# 95% bayesian credible interval
quantile(post_samples[["mu"]], 0.025)
```

```
    2.5% 
84.79651 
```

```r
quantile(post_samples[["mu"]], 0.975)
```

```
   97.5% 
88.71406 
```

Tidybayes is also very useful:

```r
fit |>
  gather_draws(mu, sigma) |>
  median_qi(.width = 0.8)
```

```
# A tibble: 2 x 7
  .variable .value .lower .upper .width .point .interval
  <chr>      <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>    
1 mu          86.8   85.4   88.0    0.8 median qi       
2 sigma       20.4   19.6   21.4    0.8 median qi       
```

Plot estimates

There are a bunch of packages, built-in functions that let you plot the estimates from the model, and I encourage you to explore these options (particularly in `bayesplot`, which we will most likely be using later on). I like using the `tidybayes` package, which allows us to easily get the posterior samples in a tidy format (e.g. using gather draws to get in long format). Once we have that, it's easy to just pipe and do ggplots as usual.

Get the posterior samples for mu and sigma in long format:

```r
dsamples <- fit  |>
  gather_draws(mu, sigma) # gather = long format
dsamples
```

```
# A tibble: 1,500 x 5
# Groups:   .variable [2]
   .chain .iteration .draw .variable .value
    <int>      <int> <int> <chr>      <dbl>
 1      1          1     1 mu          87.1
 2      1          2     2 mu          87.9
 3      1          3     3 mu          86.5
 4      1          4     4 mu          87.9
 5      1          5     5 mu          87.3
 6      1          6     6 mu          84.9
 7      1          7     7 mu          86.0
 8      1          8     8 mu          86.0
 9      1          9     9 mu          87.4
10      1         10    10 mu          86.7
# i 1,490 more rows
```

```
# wide format
fit  |>  spread_draws(mu, sigma)
```

```
# A tibble: 750 x 5
   .chain .iteration .draw    mu sigma
    <int>      <int> <int> <dbl> <dbl>
 1      1          1     1  87.1  20.1
 2      1          2     2  87.9  21.0
 3      1          3     3  86.5  20.0
 4      1          4     4  87.9  20.3
 5      1          5     5  87.3  20.9
 6      1          6     6  84.9  19.3
 7      1          7     7  86.0  19.6
 8      1          8     8  86.0  19.6
 9      1          9     9  87.4  20.8
10      1         10    10  86.7  19.7
# i 740 more rows
```

```
# quickly calculate the quantiles using

dsamples |>
  median_qi(.width = 0.8)
```

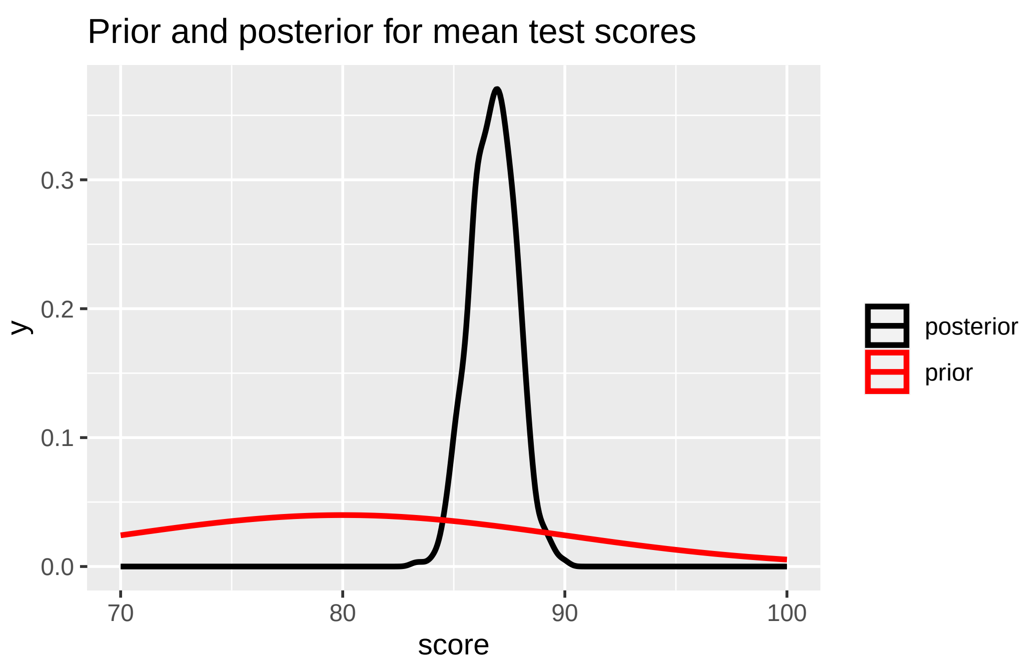
```
# A tibble: 2 x 7
```

```
  .variable .value .lower .upper .width .point .interval
  <chr>      <dbl>  <dbl>  <dbl>  <dbl> <chr>  <chr>
1 mu          86.8   85.4   88.0    0.8 median qi
2 sigma       20.4   19.6   21.4    0.8 median qi
```

Let's plot the density of the posterior samples for mu and add in the prior distribution

```r
dsamples |>
  filter(.variable == "mu") |>
  ggplot(aes(.value, color = "posterior")) + geom_density(size = 1) +
  xlim(c(70, 100)) +
  stat_function(fun = dnorm,
        args = list(mean = mu0,
                    sd = sigma0),
        aes(colour = 'prior'), size = 1) +
  scale_color_manual(name = "", values = c("prior" = "red", "posterior" = "black")) +
  ggtitle("Prior and posterior for mean test scores") +
  xlab("score")
```

```
Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
i Please use `linewidth` instead.
```



2.

Let's say we know that relationship are clear and there is little variance. We can encode this by:

```r
sigma0 <- 0.1

data <- list(y = y,
             N = length(y),
             mu0 = mu0,
             sigma0 = sigma0)
fit <- stan(file = "code/models/kids2.stan",
            data = data)
```

```
Warning in readLines(file, warn = TRUE): incomplete final line found on
'/Users/euijinbaek/STA2201/labs/code/models/kids2.stan'
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 4e-06 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.04 seconds.
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: 0.007 seconds (Warm-up)
Chain 1:                0.006 seconds (Sampling)
Chain 1:                0.013 seconds (Total)
Chain 1:
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 1e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.01 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 2: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 2: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 2: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 2: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 2: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 2: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 2: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 2: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 2: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 2: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 2: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 2:
Chain 2:  Elapsed Time: 0.007 seconds (Warm-up)
Chain 2:                0.007 seconds (Sampling)
Chain 2:                0.014 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 1e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.01 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 3: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 3: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 3: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 3: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 3: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 3: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 3: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 3: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 3: Iteration: 1600 / 2000 [ 80%]  (Sampling)
```

```
Chain 3: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 3: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.007 seconds (Warm-up)
Chain 3:                0.007 seconds (Sampling)
Chain 3:                0.014 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 1e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.01 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:    1 / 2000 [  0%]  (Warmup)
Chain 4: Iteration:  200 / 2000 [ 10%]  (Warmup)
Chain 4: Iteration:  400 / 2000 [ 20%]  (Warmup)
Chain 4: Iteration:  600 / 2000 [ 30%]  (Warmup)
Chain 4: Iteration:  800 / 2000 [ 40%]  (Warmup)
Chain 4: Iteration: 1000 / 2000 [ 50%]  (Warmup)
Chain 4: Iteration: 1001 / 2000 [ 50%]  (Sampling)
Chain 4: Iteration: 1200 / 2000 [ 60%]  (Sampling)
Chain 4: Iteration: 1400 / 2000 [ 70%]  (Sampling)
Chain 4: Iteration: 1600 / 2000 [ 80%]  (Sampling)
Chain 4: Iteration: 1800 / 2000 [ 90%]  (Sampling)
Chain 4: Iteration: 2000 / 2000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.008 seconds (Warm-up)
Chain 4:                0.007 seconds (Sampling)
Chain 4:                0.015 seconds (Total)
Chain 4:
```

Both estimates of mu and sigma are changed.

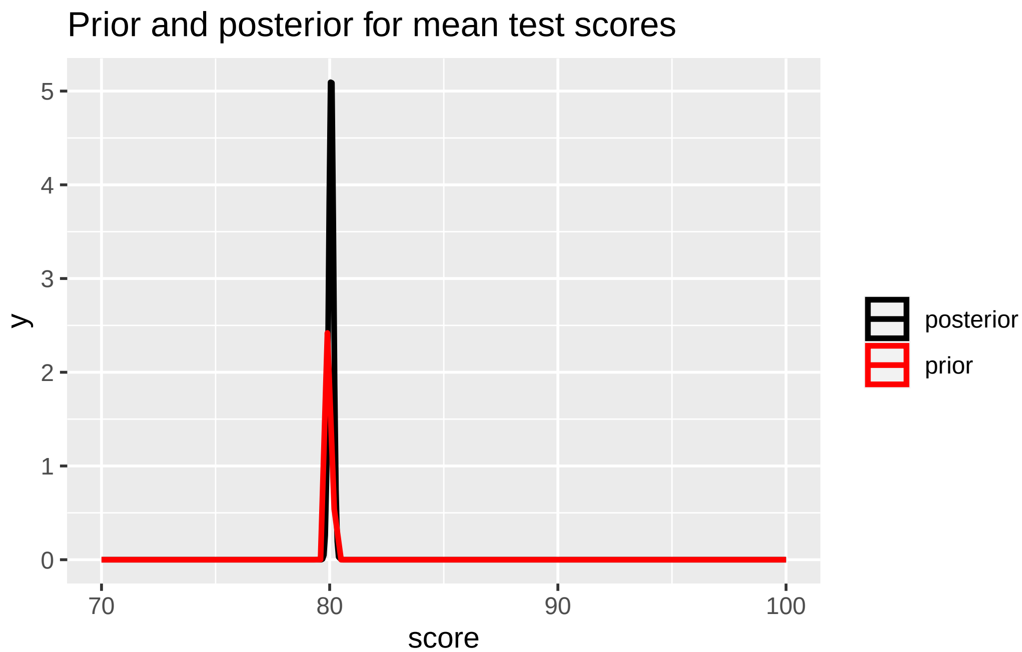
```
fit
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

          mean se_mean   sd     2.5%      25%      50%      75%    97.5% n_eff
```

```
mu       80.06    0.00 0.10    79.87    80.00    80.06    80.13    80.26  3269
sigma    21.44    0.01 0.71    20.11    20.96    21.42    21.90    22.89  3918
lp__  -1548.35    0.02 1.00 -1551.04 -1548.72 -1548.04 -1547.65 -1547.40  1696
      Rhat
mu       1
sigma    1
lp__     1

Samples were drawn using NUTS(diag_e) at Fri Feb 16 23:01:32 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Compared to the posterior when we set sigma0 $= 10$, we get distribution that has much smaller variance.

```r
# Get the posterior samples for mu and sigma
dsamples <- fit  |>
  gather_draws(mu, sigma)
# Plot
dsamples |>
  filter(.variable == "mu") |>
  ggplot(aes(.value, color = "posterior")) + geom_density(size = 1) +
  xlim(c(70, 100)) +
  stat_function(fun = dnorm,
        args = list(mean = mu0,
                    sd = sigma0),
        aes(colour = 'prior'), size = 1) +
  scale_color_manual(name = "", values = c("prior" = "red", "posterior" = "black")) +
  ggtitle("Prior and posterior for mean test scores") +
  xlab("score")
```



Adding covariates

Now let's see how kid's test scores are related to mother's education. We want to run the simple linear regression

$$y_i|\mu_i, \sigma^2 \sim N(\mu_i, \sigma^2)$$

$$\mu_i = \alpha + \beta X_i$$

Priors:

$$\alpha \sim N(0, 100^2)$$

$$\beta \sim N(0, 10^2)$$

$$\sigma \sim N(0, 10^2)$$

where $X = 1$ if the mother finished high school and zero otherwise.

`kid3.stan` has the stan model to do this. Notice now we have some inputs related to the design matrix X and the number of covariates (in this case, it's just 1).

Let's get the data we need and run the model.

```r
X <- as.matrix(kidiq$mom_hs, ncol = 1) # force this to be a matrix
K <- 1

data <- list(y = y, N = length(y),
             X =X, K = K)
fit2 <- stan(file = "code/models/kids3.stan",
            data = data,
            iter = 1000)
```

```
Warning in readLines(file, warn = TRUE): incomplete final line found on
'/Users/euijinbaek/STA2201/labs/code/models/kids3.stan'
```

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

```
Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
using C compiler: ‘Apple clang version 15.0.0 (clang-1500.0.40.1)’
using SDK: ‘MacOSX14.0.sdk’
clang -arch arm64 -I"/Library/Frameworks/R.framework/Resources/include" -DNDEBUG   -I"/Libra
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/R
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/
namespace Eigen {
^
/Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/
namespace Eigen {
               ^
               ;
In file included from <built-in>:1:
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library/RcppEigen/include/Eigen/
#include <complex>
         ^~~~~~~~~
3 errors generated.
make: *** [foo.o] Error 1
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 4.7e-05 seconds
```

```
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.47 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.071 seconds (Warm-up)
Chain 1:                0.034 seconds (Sampling)
Chain 1:                0.105 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 2.3e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.23 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 2:
Chain 2:  Elapsed Time: 0.058 seconds (Warm-up)
```

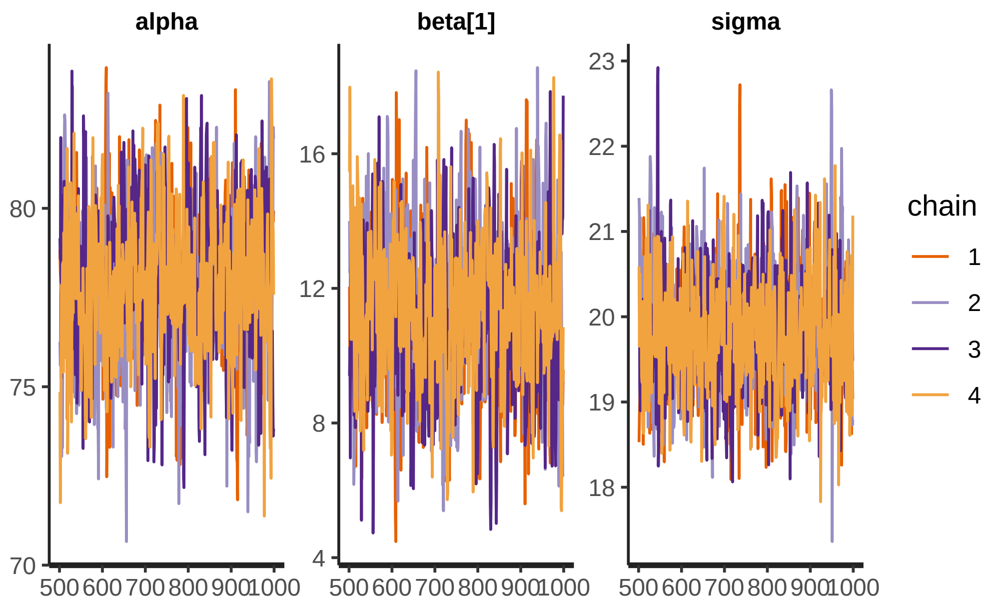
```
Chain 2:                0.04 seconds (Sampling)
Chain 2:                0.098 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 8e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.08 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.061 seconds (Warm-up)
Chain 3:                0.034 seconds (Sampling)
Chain 3:                0.095 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 1.4e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
```

```
Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.059 seconds (Warm-up)
Chain 4:                0.034 seconds (Sampling)
Chain 4:                0.093 seconds (Total)
Chain 4:
```

```
fit2
```

```
Inference for Stan model: anon_model.
4 chains, each with iter=1000; warmup=500; thin=1;
post-warmup draws per chain=500, total post-warmup draws=2000.

            mean se_mean   sd     2.5%      25%      50%      75%    97.5%
alpha      78.01    0.07 2.00    73.80    76.74    78.08    79.33    81.81
beta[1]    11.17    0.09 2.25     6.81     9.69    11.04    12.64    15.82
sigma      19.83    0.02 0.69    18.55    19.36    19.80    20.28    21.25
lp__    -1514.45    0.04 1.30 -1517.74 -1515.06 -1514.09 -1513.50 -1512.98
        n_eff Rhat
alpha     742 1.01
beta[1]   676 1.01
sigma    1222 1.00
lp__      867 1.00

Samples were drawn using NUTS(diag_e) at Fri Feb 16 23:01:54 2024.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

```
traceplot(fit2)
```



3.

Both lm() and fits show the similar coefficient of beta (slope) and alpha (intercept), which are about 11 and 77, respectively.

```
summary(fit2)
```

```
$summary
                mean     se_mean        sd        2.5%         25%         50%
alpha       78.00560 0.07361152 2.0049897   73.796549   76.74020    78.08073
beta[1]     11.16778 0.08670173 2.2539962    6.809068    9.68811    11.03930
sigma       19.83013 0.01964120 0.6864878   18.546617   19.36322    19.80321
lp__     -1514.44512 0.04399959 1.2957068 -1517.737492 -1515.06117 -1514.08784
                 75%       97.5%     n_eff      Rhat
alpha       79.32743    81.80562  741.8783 1.0058103
beta[1]     12.64357    15.81882  675.8508 1.0074985
sigma       20.27765    21.25446 1221.6021 1.0035229
lp__     -1513.49904 -1512.98014  867.1940 0.9998238

$c_summary
, , chains = chain:1

         stats
parameter        mean          sd         2.5%          25%          50%
```

```
  alpha      78.25110 1.9554547    74.024882    77.106088    78.37678
  beta[1]    10.86612 2.2320324     6.722057     9.386252    10.72092
  sigma      19.79359 0.6931693    18.502989    19.337575    19.80135
  lp__    -1514.46427 1.2856578 -1517.883717 -1515.082519 -1514.14638
         stats
parameter         75%       97.5%
  alpha      79.51041    81.70829
  beta[1]    12.26871    16.23167
  sigma      20.23059    21.31695
  lp__    -1513.51235 -1513.02410

, , chains = chain:2

         stats
parameter        mean        sd         2.5%          25%          50%
  alpha      77.75531 2.0717606    73.564393    76.44109    77.81911
  beta[1]    11.44605 2.3322571     7.068433     9.85494    11.32080
  sigma      19.86618 0.7286244    18.685036    19.34132    19.77552
  lp__    -1514.49142 1.3640802 -1517.819095 -1515.19546 -1514.05439
         stats
parameter         75%       97.5%
  alpha      79.17394    81.57956
  beta[1]    13.05164    16.13894
  sigma      20.32756    21.38115
  lp__    -1513.47694 -1512.93587

, , chains = chain:3

         stats
parameter        mean        sd         2.5%          25%          50%
  alpha      78.17288 2.0463849    73.650488    76.890560    78.32539
  beta[1]    10.93054 2.3061391     6.625693     9.400568    10.63554
  sigma      19.85895 0.6802301    18.555987    19.396084    19.84939
  lp__    -1514.47902 1.3425923 -1517.667224 -1515.063692 -1514.20720
         stats
parameter         75%       97.5%
  alpha      79.52028    81.93434
  beta[1]    12.31558    15.62599
  sigma      20.30766    21.19486
  lp__    -1513.48342 -1512.98209

, , chains = chain:4
```

```
         stats
parameter         mean        sd         2.5%        25%        50%
  alpha       77.84310 1.9033543    74.015534   76.65633   77.80818
  beta[1]     11.42840 2.0783759     7.309706   10.10653   11.38626
  sigma       19.80181 0.6399447    18.578206   19.37438   19.77750
  lp__     -1514.34576 1.1814292 -1517.442328 -1514.90053 -1513.98568
         stats
parameter          75%       97.5%
  alpha       79.00720    81.67706
  beta[1]     12.78011    15.68601
  sigma       20.20982    21.03959
  lp__     -1513.52223 -1512.97960
```

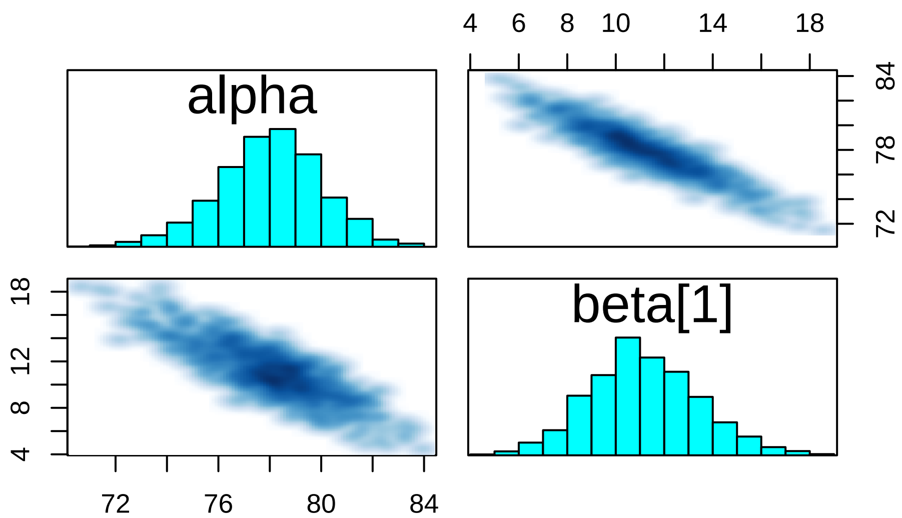
```
linear <- lm(y~kidiq$mom_hs)
summary(linear)
```

```

Call:
lm(formula = y ~ kidiq$mom_hs)

Residuals:
   Min     1Q Median     3Q    Max
-57.55 -13.32   2.68  14.68  58.45

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)    77.548      2.059  37.670  < 2e-16 ***
kidiq$mom_hs   11.771      2.322   5.069 5.96e-07 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 19.85 on 432 degrees of freedom
Multiple R-squared:  0.05613,   Adjusted R-squared:  0.05394
F-statistic: 25.69 on 1 and 432 DF,  p-value: 5.957e-07
```

It seems that they are correlated, which could be problematic. High correlation between parameters can lead to reduced sampling efficiency because we will get narrower results when sampling.

```
pairs(fit2, pars = c("alpha", "beta[1]"))
```

```
Warning in par(usr): argument 1 does not name a graphical parameter

Warning in par(usr): argument 1 does not name a graphical parameter
```

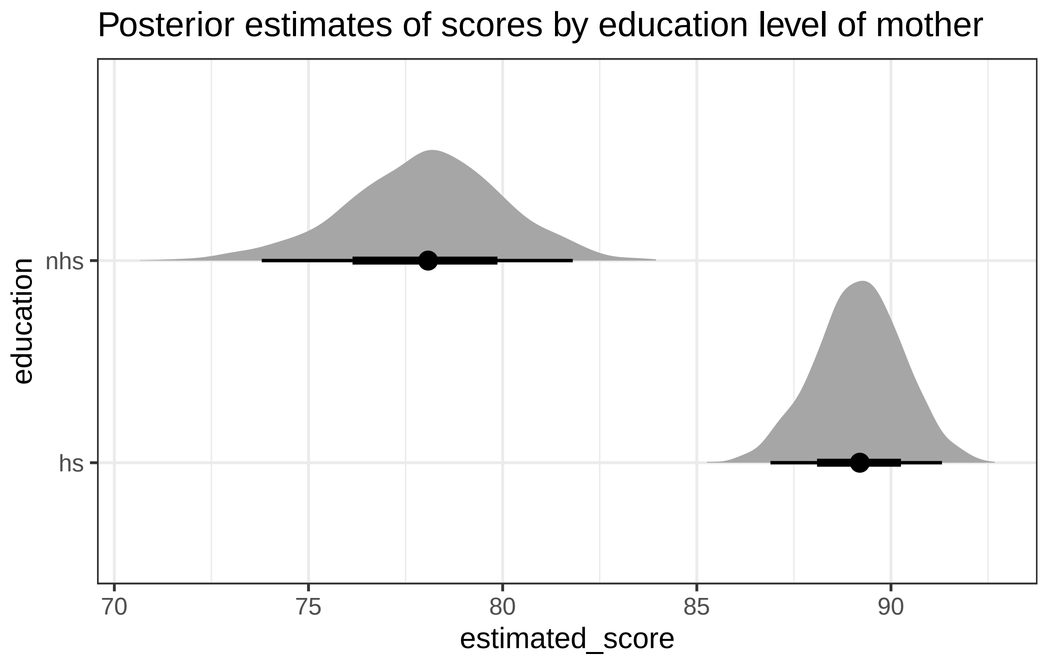


Plotting results

It might be nice to plot the posterior samples of the estimates for the non-high-school and high-school mothered kids. Here's some code that does this: notice the `beta[condition]` syntax. Also notice I'm using `spread_draws`, because it's easier to calculate the estimated effects in wide format

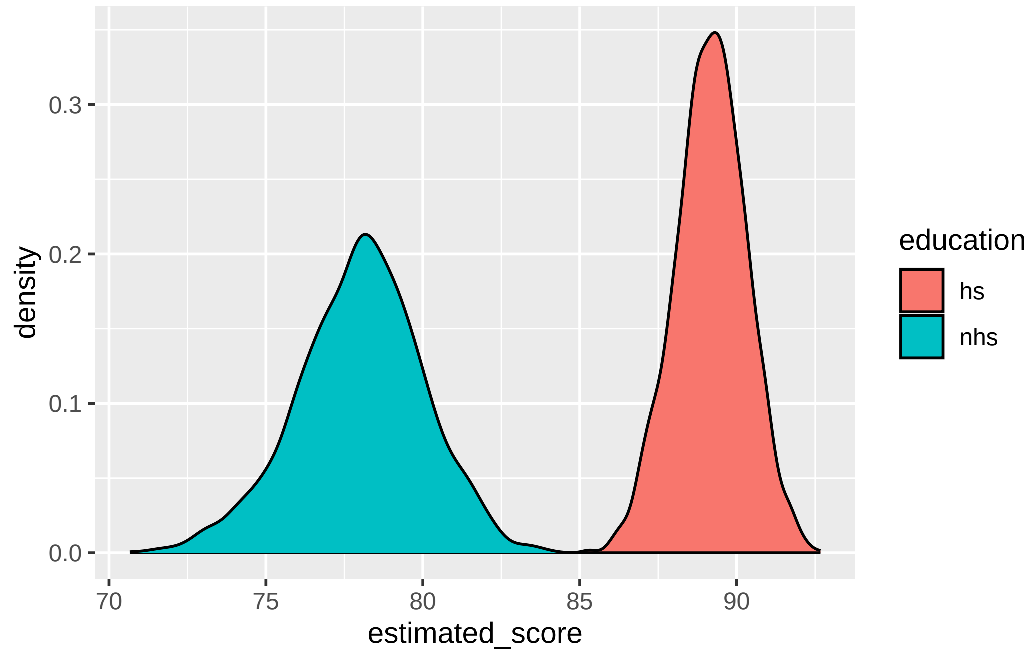
```
fit2 |>
  spread_draws(alpha, beta[k], sigma) |>
     mutate(nhs = alpha, # no high school is just the intercept
          hs = alpha + beta) |>
  select(nhs, hs) |>
  pivot_longer(nhs:hs, names_to = "education", values_to = "estimated_score") |>
  ggplot(aes(y = education, x = estimated_score)) +
  stat_halfeye() +
  theme_bw() +
  ggtitle("Posterior estimates of scores by education level of mother")
```

```
Adding missing grouping variables: `k`
```




```
fit2 |>
  spread_draws(alpha, beta[k], sigma) |>
     mutate(nhs = alpha, # no high school is just the intercept
          hs = alpha + beta) |>
  select(nhs, hs) |>
  pivot_longer(nhs:hs, names_to = "education", values_to = "estimated_score") |>
  ggplot(aes(x = estimated_score, fill = education)) +
  geom_density()
```

```
Adding missing grouping variables: `k`
```



4.

Mom's IQ has coefficient of 0.5638947, which suggest that if centered mom's IQ increases by one unit, the expected kid's test score increases about 0.56, holding all other variables constant.

```
# Centering
X <- cbind(kidiq$mom_hs, kidiq$mom_iq - mean(kidiq$mom_iq))
data <- list(y = y,
             N = length(y),
             K = 2,
             X = as.matrix(X))

fit2 <- stan(file = "code/models/kids3.stan",
            data = data,
            iter = 1000)
```

```
Warning in readLines(file, warn = TRUE): incomplete final line found on
'/Users/euijinbaek/STA2201/labs/code/models/kids3.stan'
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
```

```
Chain 1:
Chain 1: Gradient evaluation took 1.4e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.14 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1:
Chain 1:
Chain 1: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 1: Iteration: 100 / 1000 [ 10%]  (Warmup)
Chain 1: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 1: Iteration: 300 / 1000 [ 30%]  (Warmup)
Chain 1: Iteration: 400 / 1000 [ 40%]  (Warmup)
Chain 1: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 1: Iteration: 501 / 1000 [ 50%]  (Sampling)
Chain 1: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 1: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 1: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 1: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 1: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 1:
Chain 1:  Elapsed Time: 0.076 seconds (Warm-up)
Chain 1:                0.049 seconds (Sampling)
Chain 1:                0.125 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 9e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
Chain 2: Adjust your expectations accordingly!
Chain 2:
Chain 2:
Chain 2: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 2: Iteration: 100 / 1000 [ 10%]  (Warmup)
Chain 2: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 2: Iteration: 300 / 1000 [ 30%]  (Warmup)
Chain 2: Iteration: 400 / 1000 [ 40%]  (Warmup)
Chain 2: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 2: Iteration: 501 / 1000 [ 50%]  (Sampling)
Chain 2: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 2: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 2: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 2: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 2: Iteration: 1000 / 1000 [100%]  (Sampling)
```

```
Chain 2:
Chain 2:  Elapsed Time: 0.064 seconds (Warm-up)
Chain 2:                0.047 seconds (Sampling)
Chain 2:                0.111 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 9e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
Chain 3: Adjust your expectations accordingly!
Chain 3:
Chain 3:
Chain 3: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 3: Iteration: 100 / 1000 [ 10%]  (Warmup)
Chain 3: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 3: Iteration: 300 / 1000 [ 30%]  (Warmup)
Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.073 seconds (Warm-up)
Chain 3:                0.041 seconds (Sampling)
Chain 3:                0.114 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 9e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.09 seconds.
Chain 4: Adjust your expectations accordingly!
Chain 4:
Chain 4:
Chain 4: Iteration:   1 / 1000 [  0%]  (Warmup)
Chain 4: Iteration: 100 / 1000 [ 10%]  (Warmup)
Chain 4: Iteration: 200 / 1000 [ 20%]  (Warmup)
Chain 4: Iteration: 300 / 1000 [ 30%]  (Warmup)
Chain 4: Iteration: 400 / 1000 [ 40%]  (Warmup)
```

```
Chain 4: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 4: Iteration: 501 / 1000 [ 50%]  (Sampling)
Chain 4: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 4: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 4: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 4: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 4: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 4:
Chain 4:  Elapsed Time: 0.086 seconds (Warm-up)
Chain 4:                0.048 seconds (Sampling)
Chain 4:                0.134 seconds (Total)
Chain 4:
```

```
summary(fit2)
```

```
$summary
                mean     se_mean         sd          2.5%           25%
alpha     82.2185809 0.058315130 1.90075943    78.6372702    80.891933
beta[1]    5.7878903 0.067281432 2.14669309     1.6530584     4.301994
beta[2]    0.5608309 0.001692762 0.05930768     0.4424303     0.521757
sigma     18.1266018 0.015671297 0.60717049    16.9874792    17.692938
lp__   -1474.3764050 0.044850870 1.32893263 -1477.5256607 -1475.034438
                 50%           75%         97.5%     n_eff     Rhat
alpha     82.2090456    83.5316513    85.9990477 1062.4092 1.001860
beta[1]    5.7615417     7.2560090     9.9417189 1018.0039 1.001366
beta[2]    0.5619809     0.5993393     0.6749954 1227.5243 1.000146
sigma     18.1206399    18.5291665    19.3500645 1501.1063 1.000188
lp__   -1474.0980518 -1473.3890431 -1472.6403829  877.9387 1.002681

$c_summary
, , chains = chain:1

         stats
parameter          mean         sd          2.5%           25%           50%
  alpha      82.2342822 1.96592664    78.5691440    80.8178799    82.3544867
  beta[1]     5.7743382 2.17405684     1.8117494     4.2190074     5.7203888
  beta[2]     0.5624793 0.06075667     0.4433694     0.5221801     0.5607418
  sigma      18.1009895 0.58915970    16.9767825    17.7061908    18.1134285
  lp__    -1474.3845368 1.38152980 -1477.6311430 -1475.0121825 -1474.0475563
         stats
parameter           75%         97.5%
```

```
  alpha      83.6206761   86.0017929
  beta[1]     7.2640261   10.2754457
  beta[2]     0.6035686    0.6796167
  sigma      18.4685298   19.2810639
  lp__    -1473.3951592 -1472.6499914

, , chains = chain:2

         stats
parameter          mean         sd          2.5%           25%           50%
  alpha      82.3393262 1.87182973    78.7862297    81.0861029    82.3289461
  beta[1]     5.6427950 2.10781550     1.6043383     4.2300010     5.5151306
  beta[2]     0.5610832 0.05963675     0.4397319     0.5219536     0.5634447
  sigma      18.1170676 0.63904341    16.9235011    17.6739356    18.1091340
  lp__    -1474.4492122 1.41422955 -1477.5273948 -1475.2649210 -1474.1063758
         stats
parameter           75%         97.5%
  alpha      83.6317736    85.8056957
  beta[1]     7.1313234     9.8414370
  beta[2]     0.5980767     0.6763853
  sigma      18.5641672    19.4664679
  lp__    -1473.4104366 -1472.6190913

, , chains = chain:3

         stats
parameter          mean         sd         2.5%           25%           50%
  alpha      82.1117256 1.86173334    78.822843    80.7176652    81.9489162
  beta[1]     5.8776379 2.11727367     1.602707     4.5294729     6.0131734
  beta[2]     0.5580449 0.05605674     0.445697     0.5199689     0.5583854
  sigma      18.1685992 0.62619007    17.024390    17.7103536    18.1659224
  lp__    -1474.3982657 1.28601714 -1477.430316 -1475.0271094 -1474.1708834
         stats
parameter           75%         97.5%
  alpha      83.3138993    85.8885713
  beta[1]     7.3889797     9.6999316
  beta[2]     0.5963884     0.6612614
  sigma      18.5682153    19.3570495
  lp__    -1473.4816102 -1472.6464105

, , chains = chain:4

         stats
```

```
parameter          mean        sd          2.5%          25%           50%
  alpha       82.188990 1.9003756    78.5074329    80.8832050    82.2275256
  beta[1]      5.856790 2.1850645     1.7610057     4.3464607     5.8248935
  beta[2]      0.561716 0.0607382     0.4350589     0.5226205     0.5636038
  sigma       18.119751 0.5715575    17.0942632    17.6907801    18.1034402
  lp__     -1474.273605 1.2230739 -1477.3575655 -1474.9568465 -1474.0696246
         stats
parameter           75%         97.5%
  alpha      83.3791284    86.0682738
  beta[1]     7.2637145    10.0937294
  beta[2]     0.6036506     0.6735622
  sigma      18.5300877    19.3066039
  lp__    -1473.3315207 -1472.6420772
```

5.

The result agrees with 'lm()'

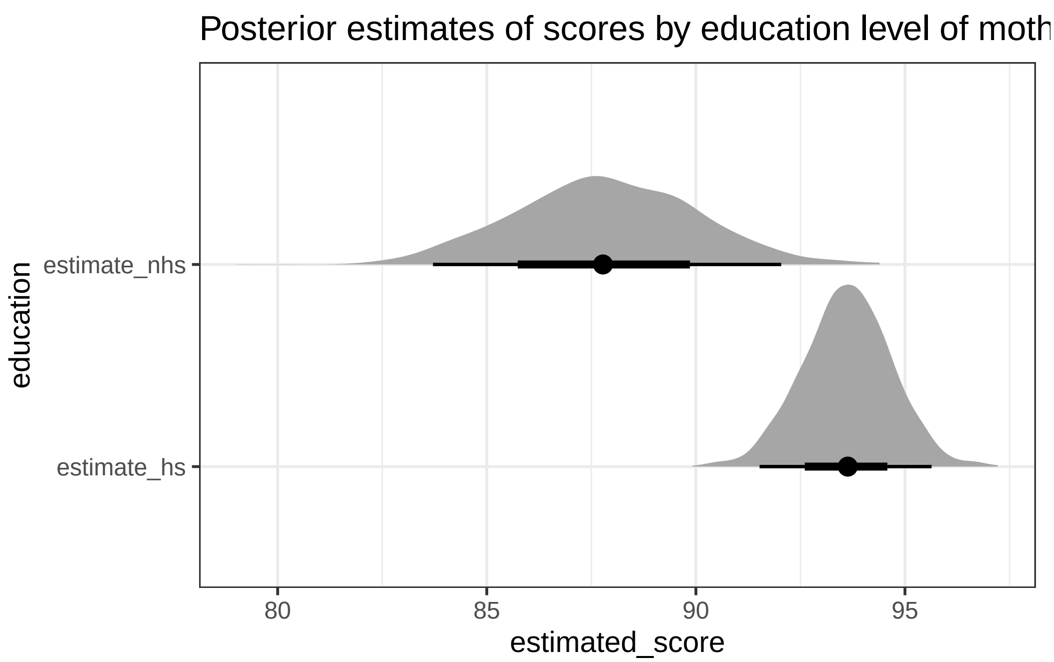
```r
linear <- lm(y ~ X[,1] +  X[,2])
summary(linear)
```

```
Call:
lm(formula = y ~ X[, 1] + X[, 2])

Residuals:
    Min      1Q  Median      3Q     Max 
-52.873 -12.663   2.404  11.356  49.545 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 82.12214    1.94370  42.250  < 2e-16 ***
X[, 1]       5.95012    2.21181   2.690  0.00742 ** 
X[, 2]       0.56391    0.06057   9.309  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 18.14 on 431 degrees of freedom
Multiple R-squared:  0.2141,	Adjusted R-squared:  0.2105 
F-statistic: 58.72 on 2 and 431 DF,  p-value: < 2.2e-16
```

6.

Plot the posterior estimates of scores by education of mother for mothers who have an IQ of 110.

```
# Using `spread_draws` to extract the relevant draws from the model.
posterior <- fit2 |>
  spread_draws(alpha, beta[k], sigma) |>
  pivot_wider(names_from = k, values_from = beta,  names_glue = "beta{k}") |>
  # Adjust for mother's IQ of 110 (centered around the mean)
  mutate(iq_adjustment = 110 - mean(kidiq$mom_iq)) |>
  mutate(estimate_nhs = alpha + iq_adjustment * beta2, # Estimate for mothers without high
         estimate_hs = alpha + iq_adjustment * beta2 + beta1) |>  # Estimate for mothers w
  select(estimate_nhs, estimate_hs) |>
  pivot_longer(estimate_nhs:estimate_hs, names_to = "education", values_to = "estimated_sc

# Plot the estimates
ggplot(posterior, aes(y = education, x = estimated_score)) +
  stat_halfeye() +
  theme_bw() +
  ggtitle("Posterior estimates of scores by education level of mother for IQ = 110")
```



7.

```
samples <- extract(fit2)
pred <- samples[["alpha"]] + samples[["beta"]][,1] + (95-mean(kidiq$mom_iq))*samples[["bet
sigma <- samples[["sigma"]]
y_pred <- tibble(y_pred = rnorm(length(sigma), mean = pred, sd = sigma))
ggplot(y_pred, aes(y_pred)) + geom_histogram(fill = "skyblue", col = "blue") + ggtitle("Di
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```