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
```

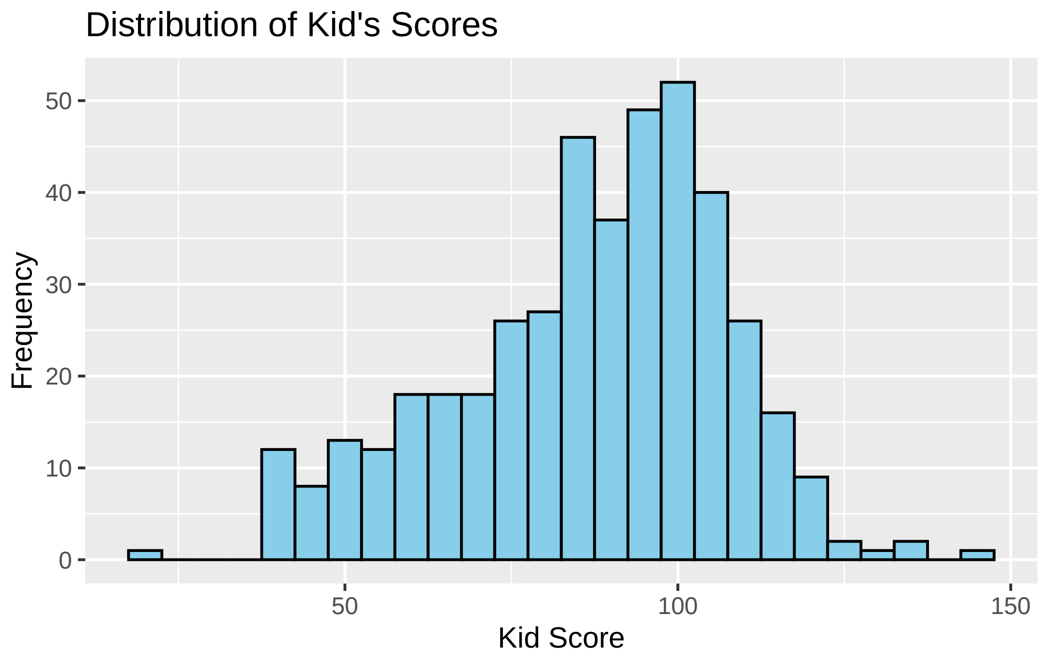
```r
# Data load
kidiq <- read_rds("data/kidiq.RDS")
head(kidiq)
```

```
# 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))
```

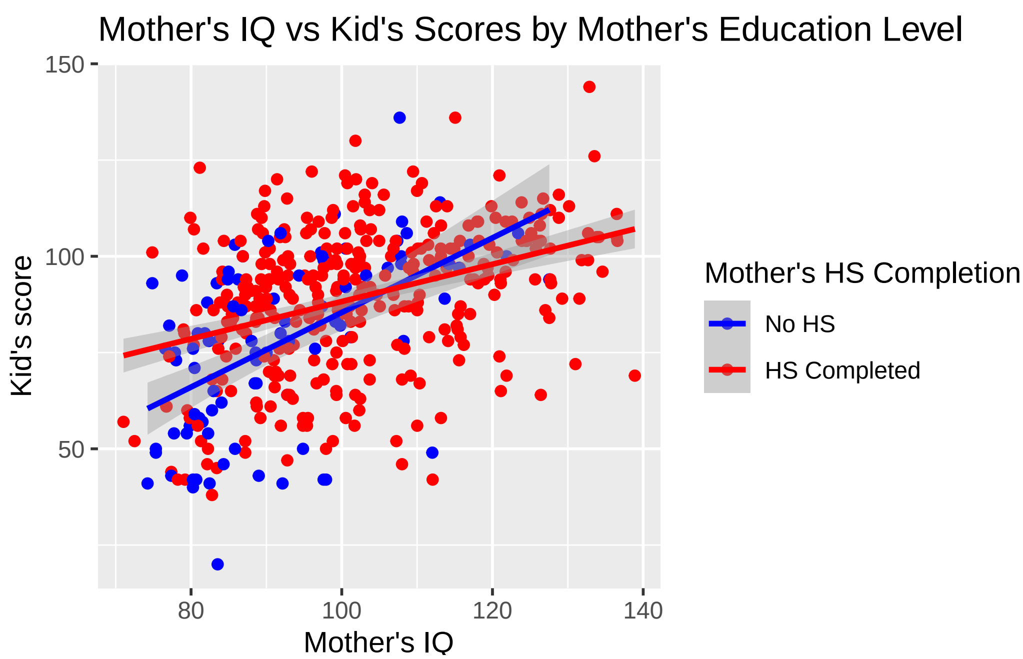
```
# A tibble: 2 x 2
  mom_hs mean_kid_score
   <dbl>          <dbl>
1      0           77.5
2      1           89.3
```

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,
```

```r
               sigma0 = sigma0)
```

Now we can run the model:

```r
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'



SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 1.7e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.17 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: 0.003 seconds (Warm-up)
Chain 1:                0.002 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 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 / 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: 0.002 seconds (Warm-up)
Chain 3:                0.001 seconds (Sampling)
Chain 3:                0.003 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
```

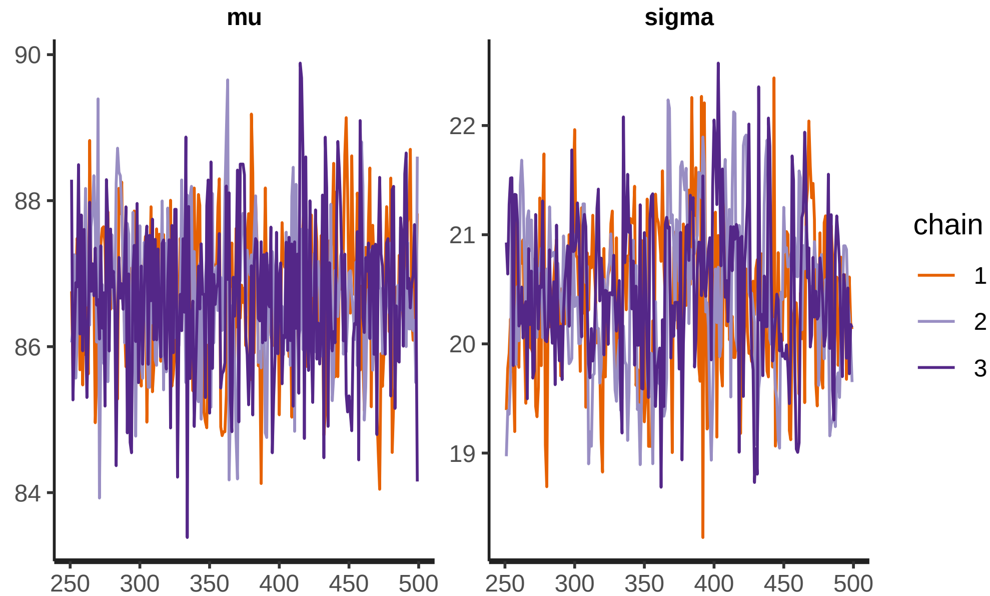
Look at the summary

```r
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      86.70    0.04 1.01    84.77    86.02    86.73    87.41    88.61   655
sigma   20.43    0.04 0.71    19.06    19.93    20.42    20.93    21.89   274
lp__ -1525.84    0.06 1.03 -1528.43 -1526.36 -1525.51 -1525.09 -1524.79   274
      Rhat
mu    1.00
sigma 1.01
lp__  1.01

Samples were drawn using NUTS(diag_e) at Fri Feb 16 04:42:34 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

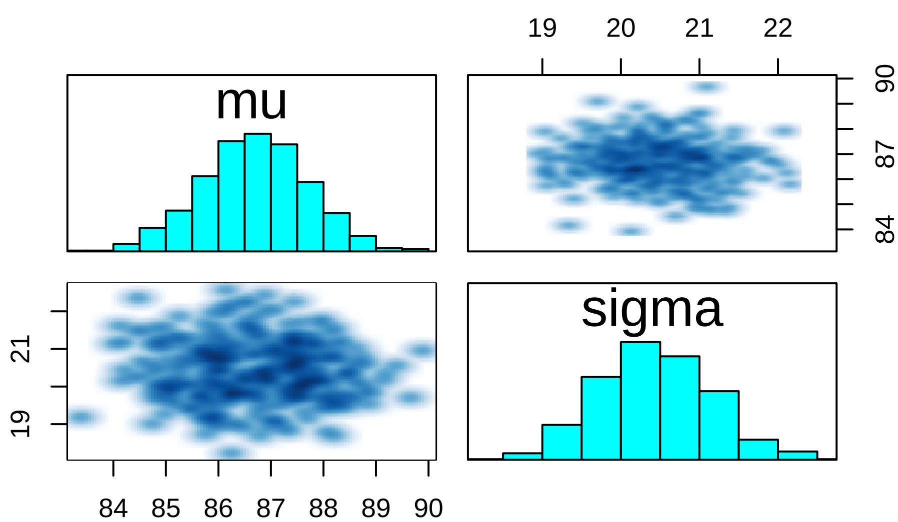
```r
traceplot(fit)
```



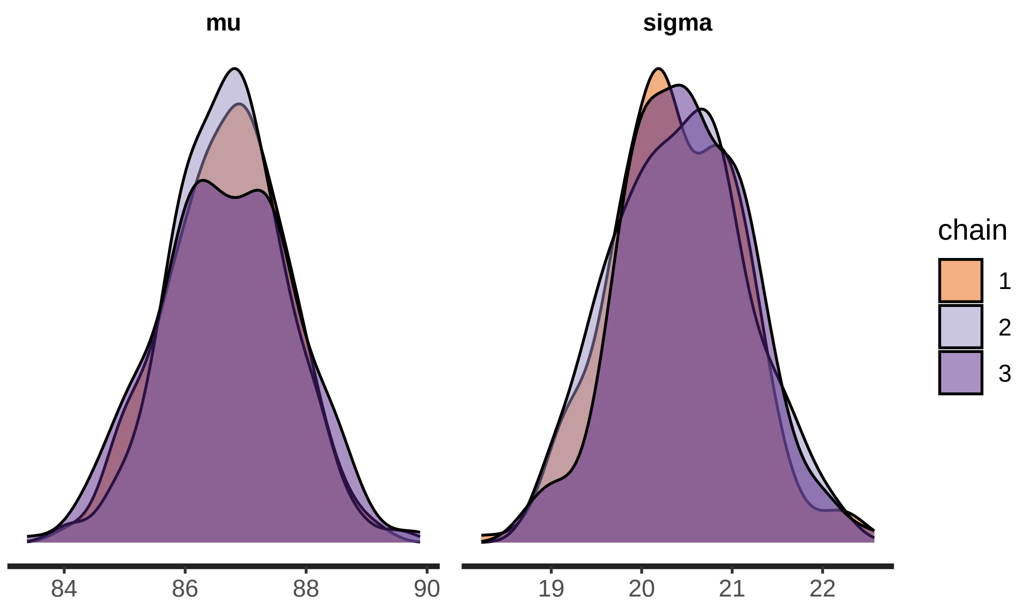
All looks fine.

```
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
```



```r
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.

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

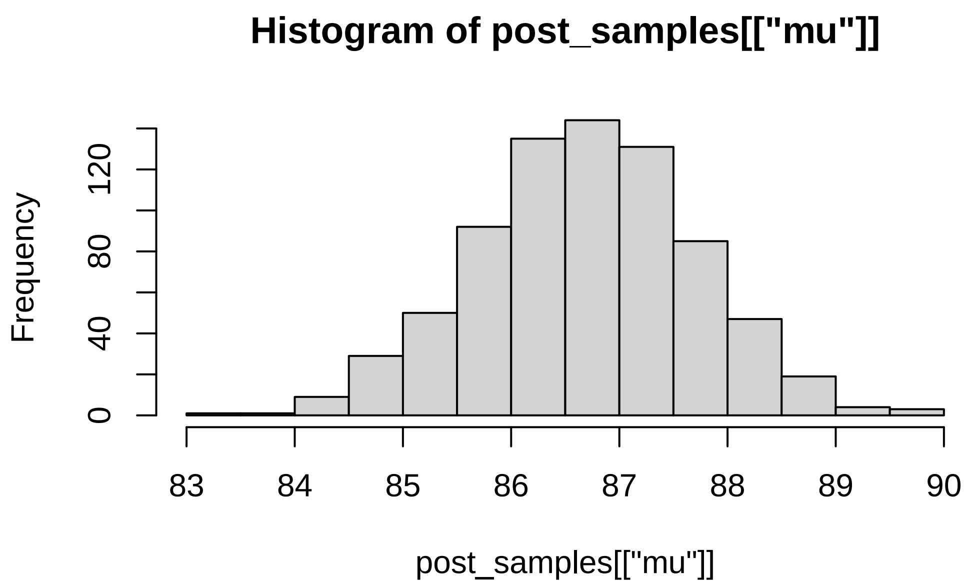
```
[1] "mu"    "sigma" "lp__"
```

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

```
[1] 87.11531 87.48351 86.51261 88.10365 85.45549 85.62602
```

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

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




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

```
[1] 86.73221
```

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

```
    2.5%
84.77055
```

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

```
   97.5%
88.60588
```

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          86.7   85.4   88.0    0.8 median qi
2 sigma       20.4   19.5   21.3    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          86.8
 2      1          2     2 mu          85.7
 3      1          3     3 mu          87.2
 4      1          4     4 mu          86.9
 5      1          5     5 mu          87.5
 6      1          6     6 mu          87.3
 7      1          7     7 mu          85.7
 8      1          8     8 mu          86.8
 9      1          9     9 mu          85.5
10      1         10    10 mu          86.9
# i 1,490 more rows
```

```r
# 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  86.8  19.4
 2      1          2     2  85.7  19.8
 3      1          3     3  87.2  19.9
 4      1          4     4  86.9  20.2
 5      1          5     5  87.5  19.8
 6      1          6     6  87.3  19.7
 7      1          7     7  85.7  19.2
 8      1          8     8  86.8  21.3
 9      1          9     9  85.5  20.2
10      1         10    10  86.9  19.8
# i 740 more rows
```

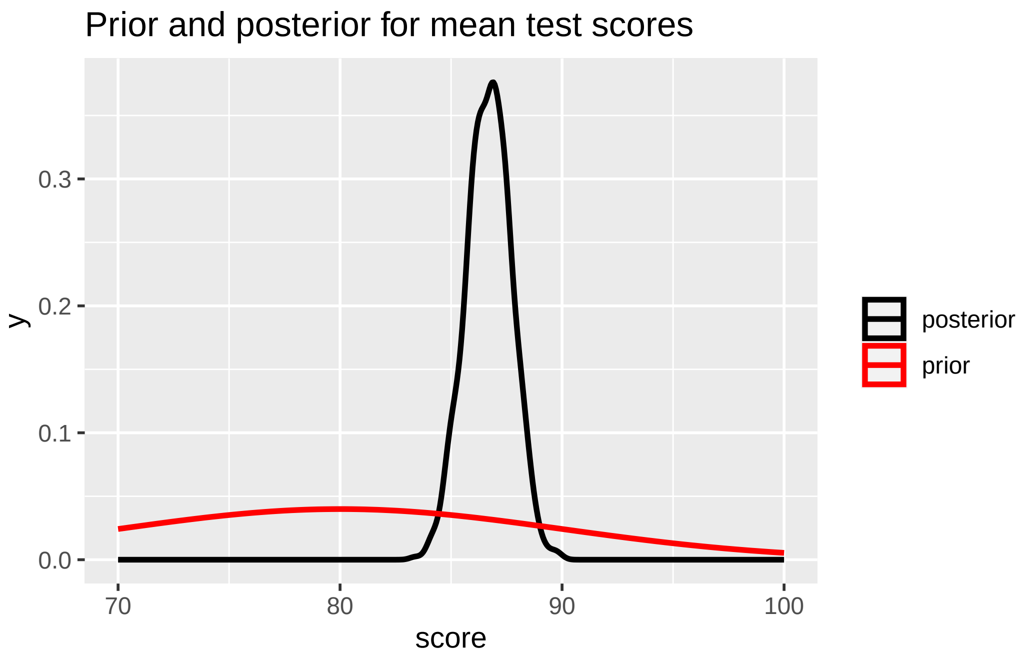
```r
# quickly calculate the quantiles using

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          86.7   85.4   88.0    0.8 median qi
2 sigma       20.4   19.5   21.3    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:

```
sigma0 <- 0.01

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: 0.008 seconds (Warm-up)
Chain 1:                0.007 seconds (Sampling)
Chain 1:                0.015 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:                0.008 seconds (Sampling)
```

```
Chain 2:                0.015 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: 0.008 seconds (Warm-up)
Chain 3:                0.007 seconds (Sampling)
Chain 3:                0.015 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: 0.01 seconds (Warm-up)
Chain 4:                0.006 seconds (Sampling)
Chain 4:                0.016 seconds (Total)
Chain 4:
```

Both estimates of mu and sigma are changed.

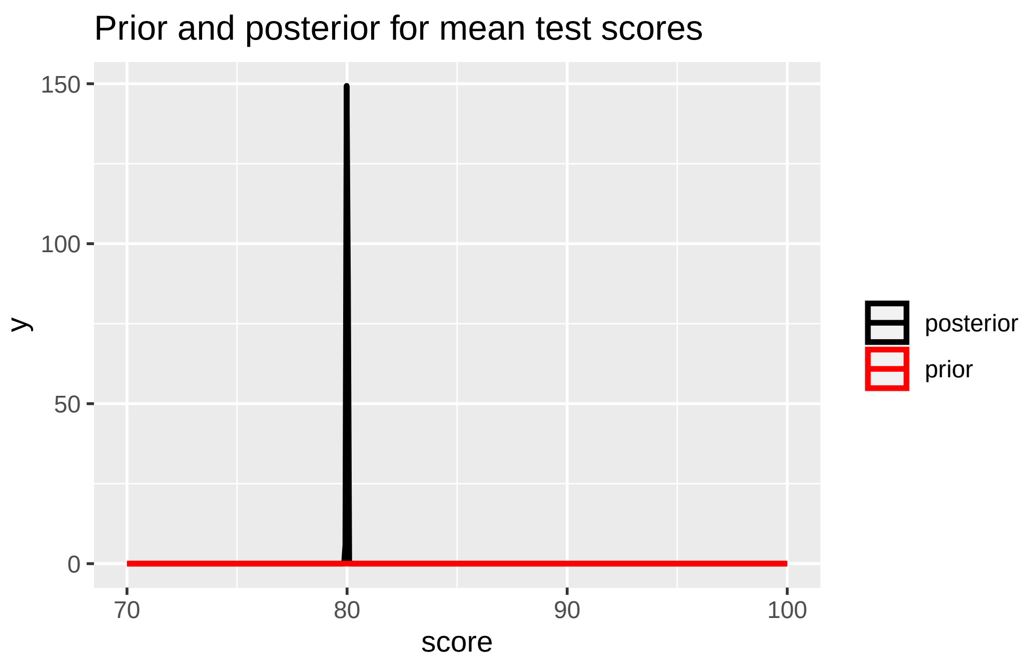
```
fit
```

```
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       80.00    0.00 0.01    79.98    79.99    80.00    80.01    80.02  3508
sigma    21.45    0.01 0.72    20.12    20.94    21.42    21.92    22.91  3942
lp__  -1548.53    0.02 0.95 -1550.99 -1548.90 -1548.24 -1547.86 -1547.59  1863
      Rhat
mu       1
sigma    1
lp__     1

Samples were drawn using NUTS(diag_e) at Fri Feb 16 04:42:36 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.

```
# Get the posterior samples for mu and sigma
dsamples <- fit  |>
  gather_draws(mu, sigma)
# Plot
dsamples |>
  filter(.variable == "mu") |>
```

```r
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'


SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 4.8e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.48 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:                0.03 seconds (Sampling)
Chain 1:                0.101 seconds (Total)
Chain 1: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2: 
Chain 2: Gradient evaluation took 8e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.08 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: 0.068 seconds (Warm-up)
Chain 2:                0.039 seconds (Sampling)
Chain 2:                0.107 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)
```

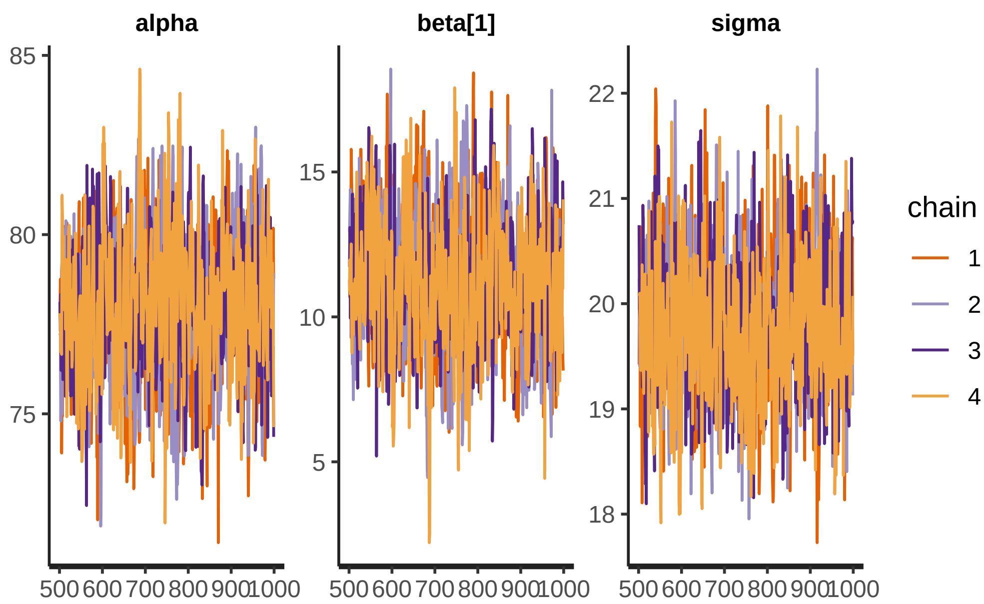
```
Chain 3: Iteration: 400 / 1000 [ 40%]  (Warmup)
Chain 3: Iteration: 500 / 1000 [ 50%]  (Warmup)
Chain 3: Iteration: 501 / 1000 [ 50%]  (Sampling)
Chain 3: Iteration: 600 / 1000 [ 60%]  (Sampling)
Chain 3: Iteration: 700 / 1000 [ 70%]  (Sampling)
Chain 3: Iteration: 800 / 1000 [ 80%]  (Sampling)
Chain 3: Iteration: 900 / 1000 [ 90%]  (Sampling)
Chain 3: Iteration: 1000 / 1000 [100%]  (Sampling)
Chain 3:
Chain 3:  Elapsed Time: 0.058 seconds (Warm-up)
Chain 3:                0.037 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 8e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.08 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: 0.061 seconds (Warm-up)
Chain 4:                0.035 seconds (Sampling)
Chain 4:                0.096 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      77.88    0.07 1.97   73.97   76.56   77.90   79.27   81.72
beta[1]    11.33    0.09 2.26    6.99    9.81   11.28   12.88   15.78
sigma      19.81    0.02 0.68   18.49   19.35   19.80   20.25   21.18
lp__    -1514.35    0.05 1.20 -1517.44 -1514.83 -1514.06 -1513.47 -1512.97
        n_eff Rhat
alpha     755    1
beta[1]   703    1
sigma    1075    1
lp__      596    1

Samples were drawn using NUTS(diag_e) at Fri Feb 16 04:42:57 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.

```r
summary(fit2)
```

```
$summary
               mean    se_mean        sd         2.5%          25%          50%
alpha      77.88247 0.07156095 1.9661078    73.969177    76.560489     77.90160
beta[1]    11.32504 0.08523758 2.2604708     6.991923     9.810953     11.27511
sigma      19.80749 0.02069764 0.6787321    18.486643    19.350864     19.79917
lp__    -1514.34615 0.04894879 1.1950312 -1517.443918 -1514.830410 -1514.05907
                75%       97.5%     n_eff     Rhat
alpha      79.27294    81.72349  754.8532 1.003429
beta[1]    12.87935    15.77861  703.2918 1.002784
sigma      20.25457    21.17824 1075.3628 1.004450
lp__    -1513.47150 -1512.97472  596.0387 1.002691

$c_summary
, , chains = chain:1

         stats
parameter        mean        sd         2.5%          25%         50%
  alpha      77.62104 2.0618638    73.652251    76.192080    77.62494
  beta[1]    11.59264 2.3478271     7.192052     9.875202    11.67965
  sigma      19.84839 0.7222719    18.433160    19.367964    19.84204
  lp__    -1514.48005 1.2329611 -1517.773720 -1515.118880 -1514.18958
         stats
parameter         75%       97.5%
  alpha      79.20157    81.16011
  beta[1]    13.21873    15.97551
  sigma      20.35315    21.22781
  lp__    -1513.54525 -1513.00328

, , chains = chain:2

         stats
parameter        mean        sd         2.5%          25%         50%
  alpha      77.86925 1.9787880    73.989413    76.572877    77.85859
  beta[1]    11.31729 2.3225816     6.760015     9.857272    11.31108
  sigma      19.84386 0.6501825    18.593368    19.420660    19.80577
```

```
  lp__    -1514.38040 1.2389854 -1517.576708 -1514.871059 -1514.15424
         stats
parameter         75%       97.5%
  alpha      79.23090    82.08289
  beta[1]    12.99241    15.80302
  sigma      20.23083    21.19708
  lp__    -1513.46922 -1512.97396

, , chains = chain:3

         stats
parameter        mean        sd         2.5%          25%         50%
  alpha      78.01686 1.7752274    74.130079    76.844167    78.08046
  beta[1]    11.18501 2.0352490     7.541631     9.816764    11.06441
  sigma      19.82598 0.6671992    18.635949    19.349526    19.85022
  lp__    -1514.21473 1.0522406 -1516.889152 -1514.657562 -1513.97423
         stats
parameter         75%       97.5%
  alpha      79.27339    81.35092
  beta[1]    12.51116    15.71540
  sigma      20.26783    21.14601
  lp__    -1513.45648 -1512.96748

, , chains = chain:4

         stats
parameter        mean        sd         2.5%          25%         50%
  alpha      78.02273 2.0151368    74.158562    76.597815    78.10328
  beta[1]    11.20523 2.3054008     6.414633     9.756408    11.14684
  sigma      19.71172 0.6658642    18.401371    19.300533    19.69098
  lp__    -1514.30941 1.2331654 -1517.475154 -1514.756098 -1513.99105
         stats
parameter         75%       97.5%
  alpha      79.33643    82.50294
  beta[1]    12.86571    15.51217
  sigma      20.13838    21.01954
  lp__    -1513.42981 -1512.94948
```

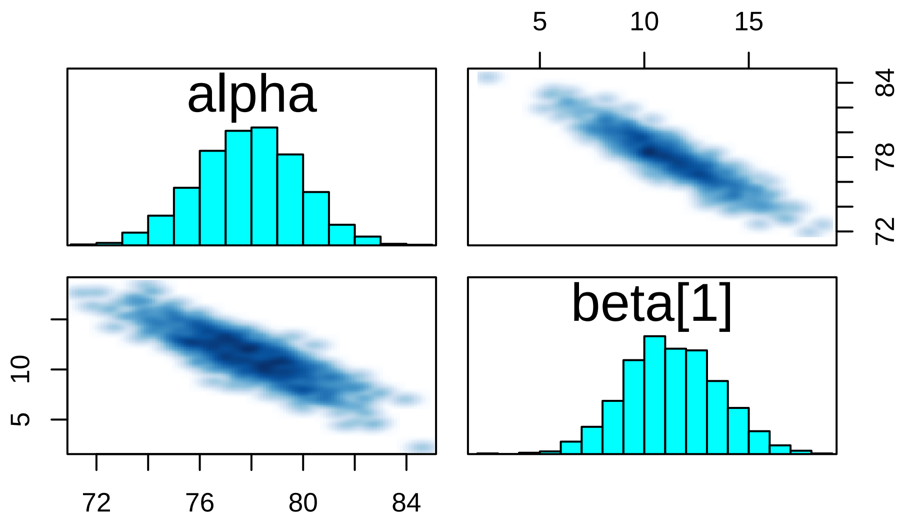
```r
linear <- lm(y~kidiq$mom_hs)
summary(linear)
```

```
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.

```r
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
```

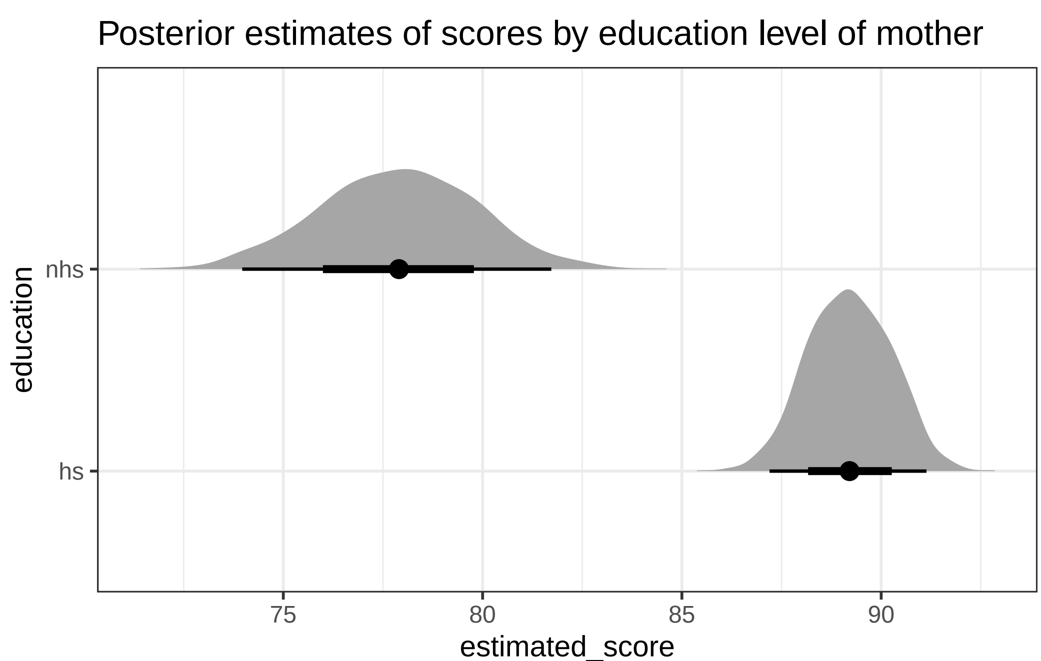


Plotting results

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

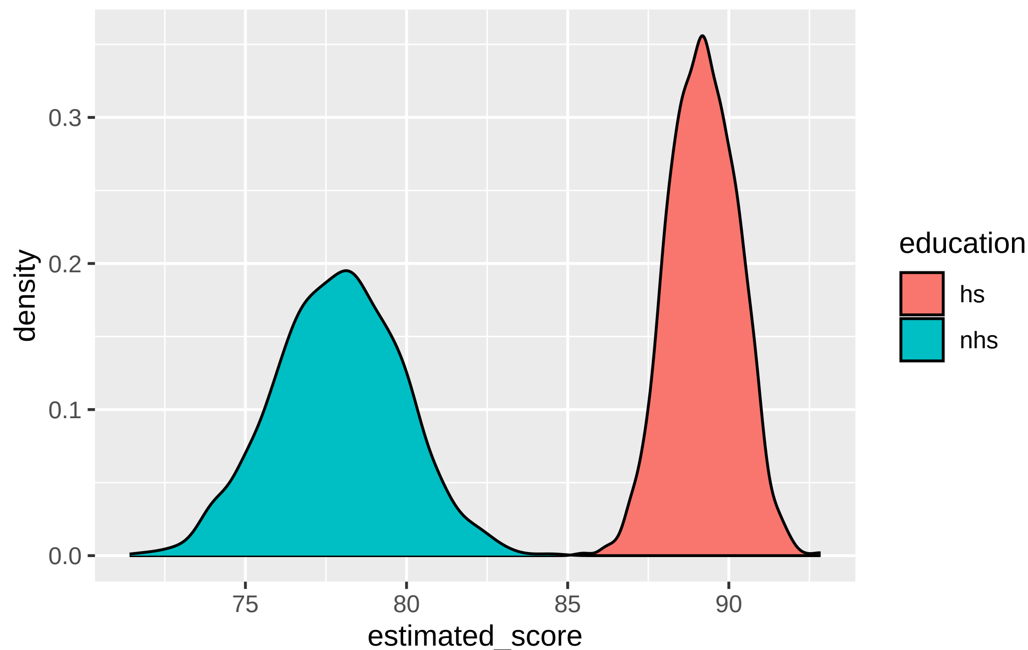
```
fit2 |>
  spread_draws(alpha, beta[k], sigma) |>
     mutate(nhs = alpha, # no high school is just the intercept
          hs = alpha + beta) |>
  select(nhs, hs) |>
  pivot_longer(nhs:hs, names_to = "education", values_to = "estimated_score") |>
  ggplot(aes(y = education, x = estimated_score))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

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



```r
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.

```r
# Centering
X <- cbind(kidiq$mom_hs, kidiq$mom_iq - mean(kidiq$mom_iq))
data <- list(y = y,
             N = length(y),
```

```r
               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 1.5e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.15 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: 0.064 seconds (Warm-up)
Chain 1:                0.041 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 8e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.08 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:                0.046 seconds (Sampling)
Chain 2:                0.11 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: 0.061 seconds (Warm-up)
Chain 3:                0.047 seconds (Sampling)
Chain 3:                0.108 seconds (Total)
```

```
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 8e-06 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.08 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: 0.072 seconds (Warm-up)
Chain 4:                0.037 seconds (Sampling)
Chain 4:                0.109 seconds (Total)
Chain 4:
```

```r
summary(fit2)
```

```
$summary
                 mean     se_mean         sd          2.5%           25%
alpha      82.3596794 0.065167009 1.88647418    78.6369001    81.0827747
beta[1]     5.6478550 0.076184747 2.14527826     1.4687316     4.2184405
beta[2]     0.5650369 0.001649101 0.06042873     0.4484414     0.5240641
sigma      18.1383962 0.017043484 0.62009010    16.9086903    17.7316013
lp__    -1474.4196569 0.046298929 1.38726341 -1477.9356359 -1475.1610045
                  50%           75%         97.5%     n_eff     Rhat
alpha      82.3842472    83.6093916    86.0740513  838.0041 1.000690
beta[1]     5.6757784     7.0527118     9.8831565  792.9233 1.001820
beta[2]     0.5628539     0.6066341     0.6859341 1342.7418 1.000283
sigma      18.1146879    18.5434820    19.3164283 1323.7099 1.000210
```

```
lp__    -1474.1276433 -1473.3702942 -1472.6601842  897.7925 1.002588

$c_summary
, , chains = chain:1

         stats
parameter          mean         sd          2.5%           25%           50%
  alpha      82.4657300 1.92208048    78.6059338    81.1636151    82.4392803
  beta[1]     5.4778735 2.15497372     1.2952542     4.0453456     5.4600873
  beta[2]     0.5687391 0.05881919     0.4634682     0.5259723     0.5677915
  sigma      18.1231004 0.68242558    16.7803087    17.6593877    18.1137566
  lp__    -1474.5179842 1.43118637 -1477.9481708 -1475.2255858 -1474.2634724
         stats
parameter           75%         97.5%
  alpha      83.7126019    86.1099453
  beta[1]     6.8341378     9.7635075
  beta[2]     0.6112038     0.6833191
  sigma      18.5972366    19.3783453
  lp__    -1473.4537833 -1472.6667117

, , chains = chain:2

         stats
parameter          mean         sd          2.5%           25%           50%
  alpha      82.3721866 1.87848000    78.8051583    81.1060410    82.3649485
  beta[1]     5.6537891 2.11672040     1.4703487     4.3144907     5.7310332
  beta[2]     0.5632653 0.06466778     0.4379153     0.5235077     0.5615896
  sigma      18.1048730 0.56977415    17.0247270    17.7496721    18.1022529
  lp__    -1474.4189376 1.39013129 -1478.0697831 -1475.0959384 -1474.1169068
         stats
parameter           75%         97.5%
  alpha      83.5648780    86.2156369
  beta[1]     7.1526490     9.6083111
  beta[2]     0.6088563     0.6903906
  sigma      18.4395676    19.2574602
  lp__    -1473.3683199 -1472.6377862

, , chains = chain:3

         stats
parameter          mean         sd         2.5%           25%           50%
  alpha      82.2637360 1.88615239    78.473144    81.0034776    82.4187831
  beta[1]     5.8240544 2.10943684     1.810397     4.4100815     5.7341319
```

```
  beta[2]      0.5603897 0.05670269      0.448546      0.5234739      0.5585313
  sigma       18.1802801 0.61500149     17.059401     17.7433737     18.1482175
  lp__     -1474.4049866 1.40810376 -1478.086930 -1475.1632076 -1474.0853790
         stats
parameter            75%         97.5%
  alpha       83.5633273    85.5874996
  beta[1]      7.0146565    10.1789213
  beta[2]      0.5959998     0.6775045
  sigma       18.6112305    19.3746662
  lp__     -1473.3239904 -1472.6517059

, , chains = chain:4

         stats
parameter           mean         sd          2.5%           25%           50%
  alpha       82.3370651 1.85868810    78.8461902    81.0664727    82.2970718
  beta[1]      5.6357031 2.19138308     0.9838831     4.1186922     5.7067503
  beta[2]      0.5677534 0.06103893     0.4486392     0.5247857     0.5647654
  sigma       18.1453314 0.60713924    17.1063051    17.7502207    18.1143999
  lp__     -1474.3367193 1.31501230 -1477.5565446 -1475.1352232 -1474.0348279
         stats
parameter            75%         97.5%
  alpha       83.5792093    86.1270585
  beta[1]      7.2009813     9.6965209
  beta[2]      0.6088205     0.6873818
  sigma       18.5274205    19.3377683
  lp__     -1473.3347024 -1472.7088693
```

5.

The result agrees with 'lm()'

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

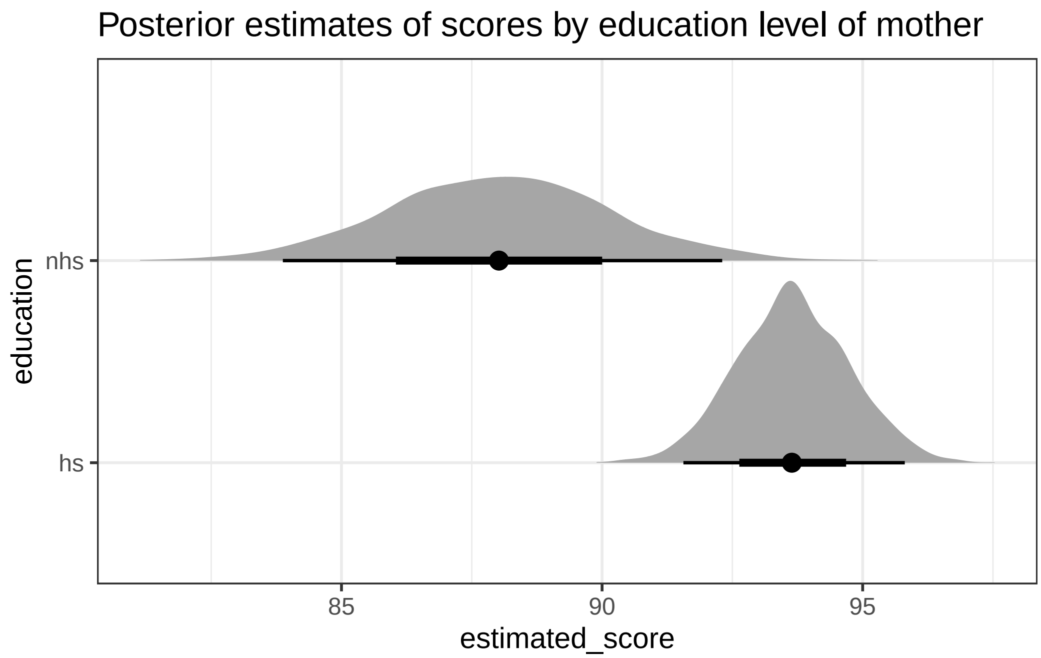
```
Call:
lm(formula = y ~ X[, 1] + X[, 2])

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.

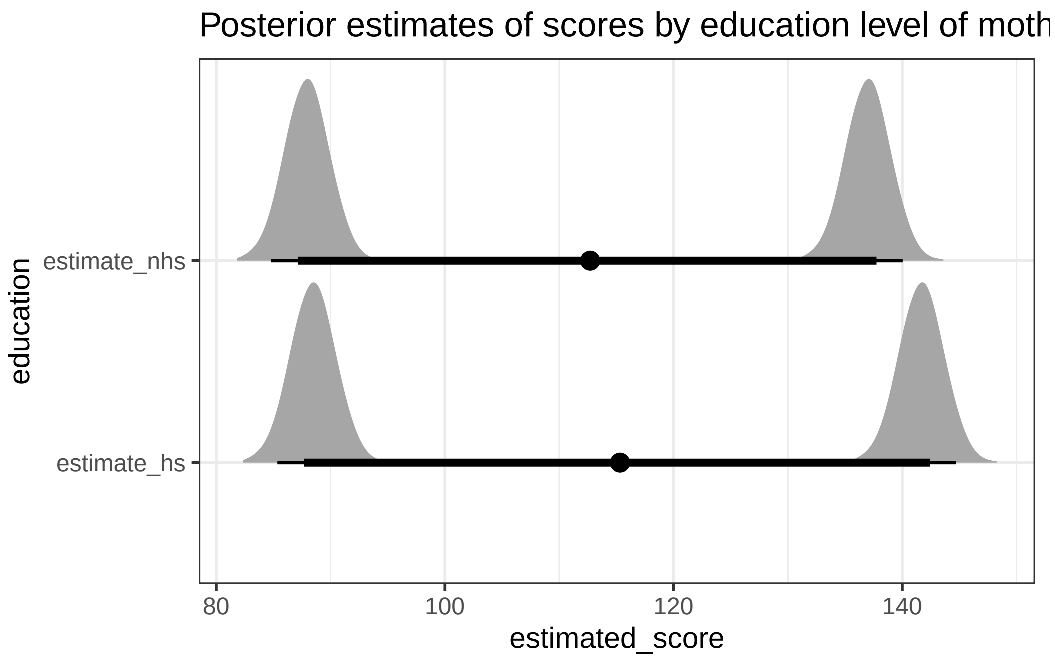
```r
fit2 |>
  spread_draws(alpha, beta[k], sigma) |>
  pivot_wider(names_from = k, values_from = beta,  names_glue = "beta{k}") |>
  mutate(nhs=alpha+beta2*(110-mean(kidiq$mom_iq)),
         hs=alpha+beta1+beta2*(110-mean(kidiq$mom_iq)))|>
  select(nhs, hs) |>
  pivot_longer(nhs:hs, names_to = "education", 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")
```




```
# Using `spread_draws` to extract the relevant draws from the model.
posterior <- fit2 |>
  spread_draws(alpha, beta[k], sigma) |>
  # 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 * beta[2], # Estimate for mothers without hi
         estimate_hs = alpha + iq_adjustment * beta[2] + beta[1]) %>%  # Estimate for moth
  select(estimate_nhs, estimate_hs) |>
  pivot_longer(estimate_nhs:estimate_hs, names_to = "education", values_to = "estimated_sc
```

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

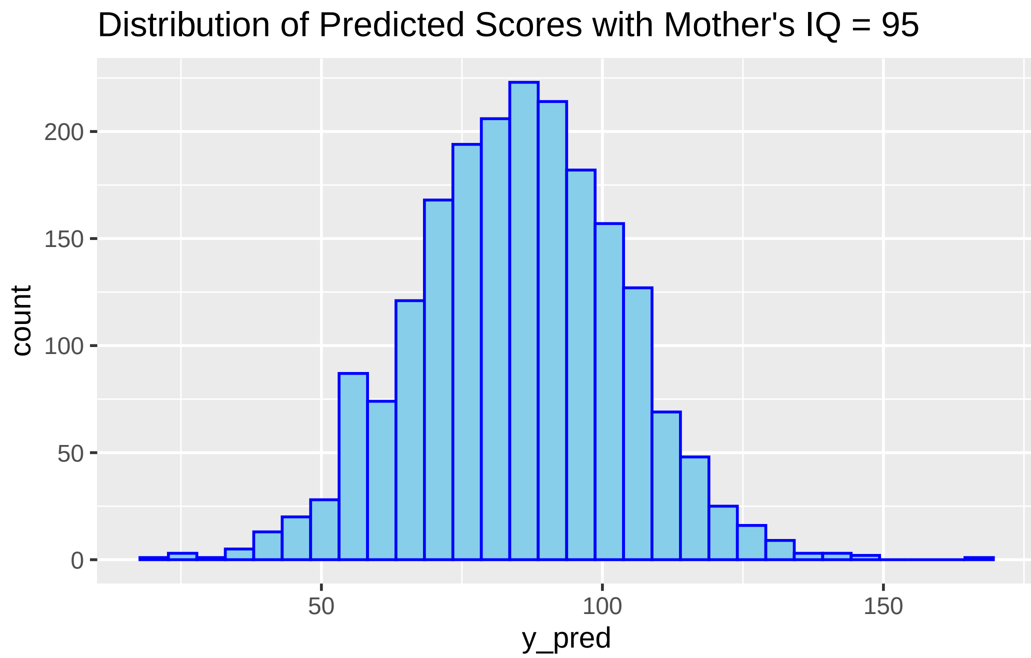
```
# Plot the estimates
ggplot(posterior, aes(y = education, x = estimated_score)) +
  stat_halfeye() +
  theme_bw() +
  ggtitle("Posterior estimates of scores by education level of mother for IQ = 110")
```



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`.
```


Distribution of Predicted Scores with Mother's IQ = 95
count
200
150
100
50
0
50
100
150
y_pred