Week 5: Bayesian linear regression and introduction to Stan

16/02/24

Introduction

Today we will be starting off using Stan, looking at the kid's test score data set (available in resources for the Gelman Hill textbook).

```
#install.packages('rstan')
#install.packages('tidyverse')
#install.packages('tidybayes')
#install.packages('here')
#install.packages('skimr')
library(tidyverse)
library(rstan)
library(tidybayes)
library(here)
library(skimr)
```

The data look like this:

```
kidiq <- read_rds(here("kidiq.RDS"))
kidiq
```

```
# A tibble: 434 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
 7        69      1  139.       20
 8       106      1  125.       23
 9       102      1   81.6      24
10        95      1   95.1      19
# i 424 more rows
```

As well as the kid's test scores, we have a binary variable indicating whether or not the mother completed high school, the mother's IQ and age.

Descriptives

Question 1

Use plots or tables to show three interesting observations about the data. Remember:

- Explain what your graph/ tables show
- Choose a graph type that's appropriate to the data type

```
skim(kidiq)
```

Table 1: Data summary

Name	kidiq
Number of rows	434
Number of columns	4
Column type frequency:	
numeric	4
Group variables	None

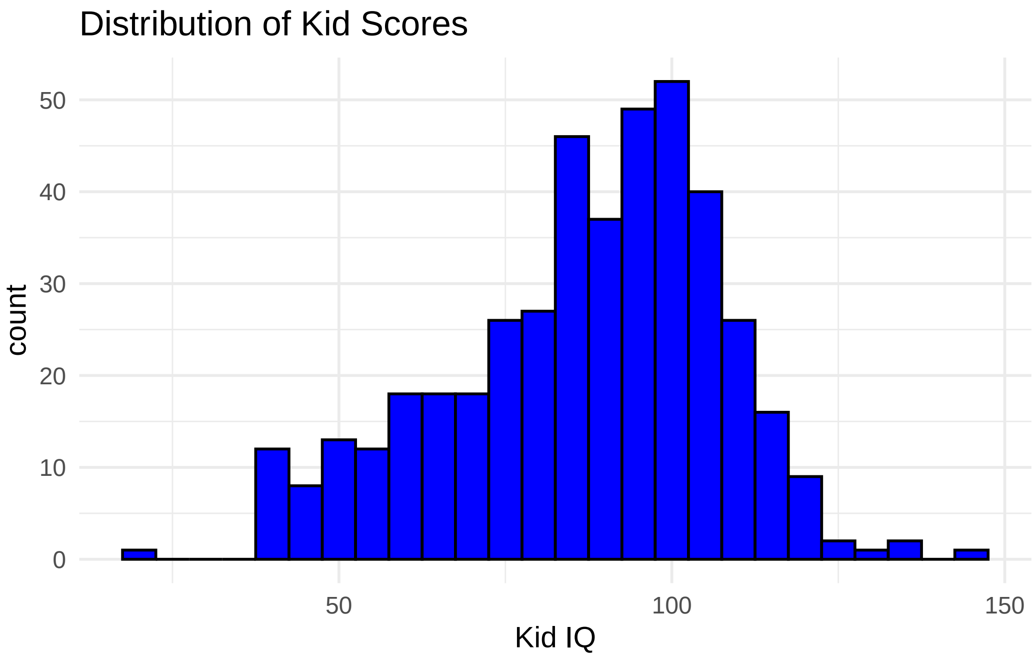
Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
kid_score	0	1	86.80	20.41	20.00	74.00	90.00	102.00	144.00	
mom_hs	0	1	0.79	0.41	0.00	1.00	1.00	1.00	1.00	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
mom_iq	0	1	100.00	15.00	71.04	88.66	97.92	110.27	138.89	
mom_age	0	1	22.79	2.70	17.00	21.00	23.00	25.00	29.00	

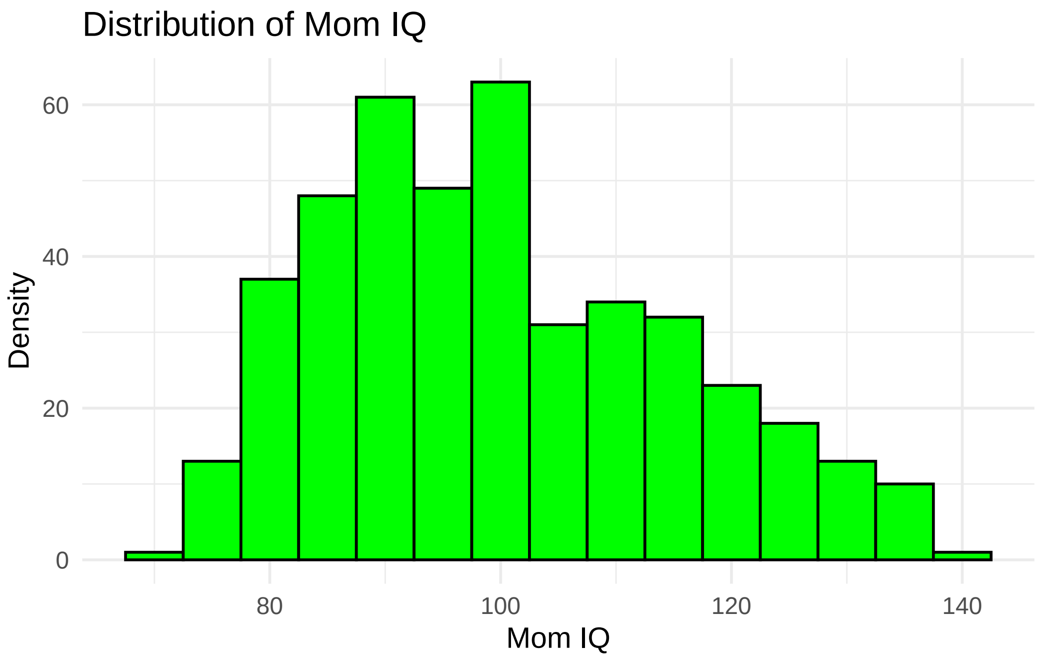
From dataset, we know that the kid_score, mom_iq, mom_age are numerical variable and the mom_hs is a factor. There are no missing values in the dataset. For kid_score with a mean score of approximately 86.797 and a standard deviation of about 20.41. The 25th percentile is 74, and the median (50th percentile) is 90.For mom_iq, with mean of 100, standard deviation of 15, and percentiles indicating a normal distribution around the mean value. The mean age of mom is around 22.79 years and a standard deviation of approximately 2.70. The median age is 23 and 75th percentile is 25, which indicates the age of mom is young.

```r
kidiq |>
  ggplot(aes(kid_score)) +
  geom_histogram(binwidth = 5, fill = "blue", color = "black") +
  labs(title = "Distribution of Kid Scores", x = "Kid IQ") +
  theme_minimal()
```

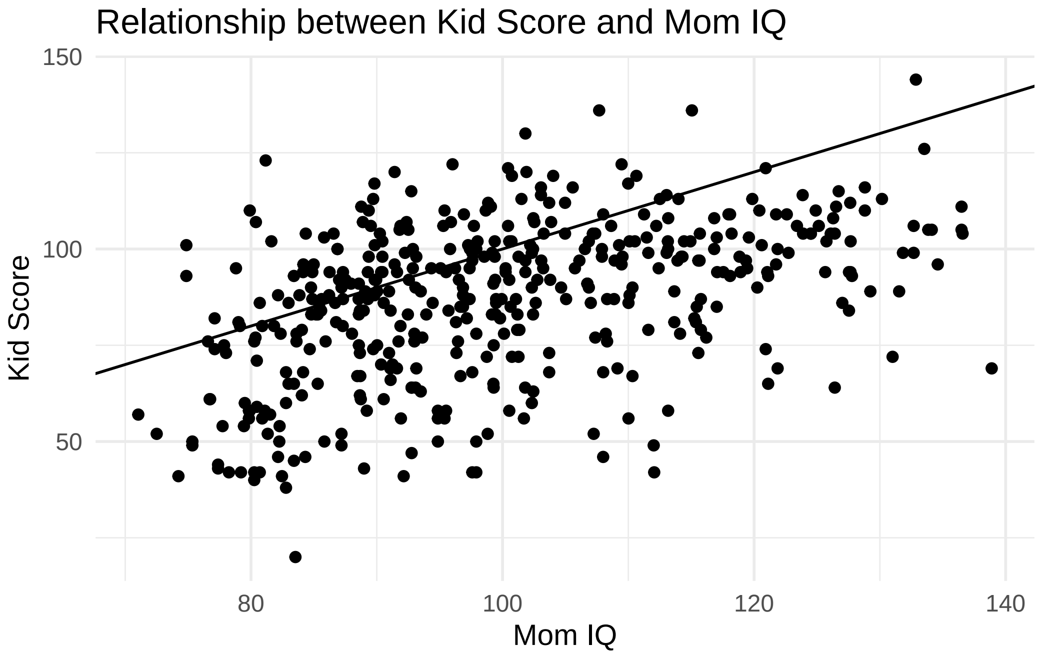


The histogram illustrates the distribution of Kid's IQ. The x-axis represents the range of IQ, and the y-axis represents the count of IQ. The histogram shows a unimodal distribution with the highest concentration of IQ around 100. The distribution appears to be symmetric around this peak, suggesting a normal distribution of IQ. The scores extend from near 0 to 150 approximately, but there are very few IQs at these extremes which matches the result of table.

```r
kidiq |>
  ggplot(aes(mom_iq)) +
  geom_histogram(binwidth = 5, fill = "green", color = "black") +
  labs(title = "Distribution of Mom IQ", x = "Mom IQ", y = "Density") +
  theme_minimal()
```

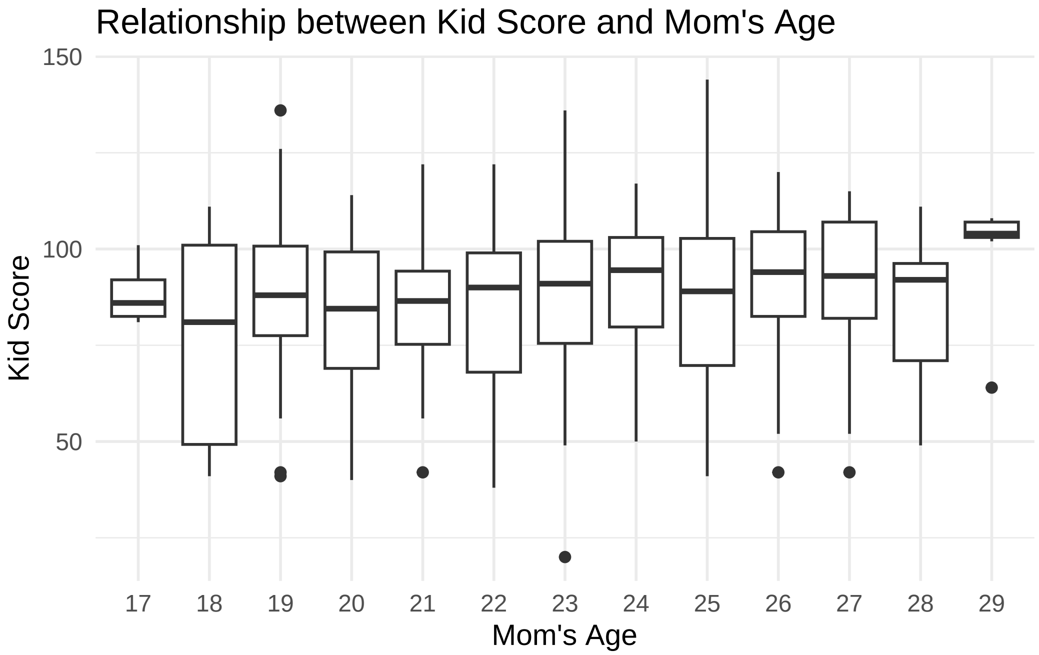


```r
kidiq |>
  ggplot(aes(mom_iq, kid_score)) +
  geom_point() +
  geom_abline()+
  labs(title = "Relationship between Kid Score and Mom IQ", x = "Mom IQ", y = "Kid Score")
  theme_minimal()
```



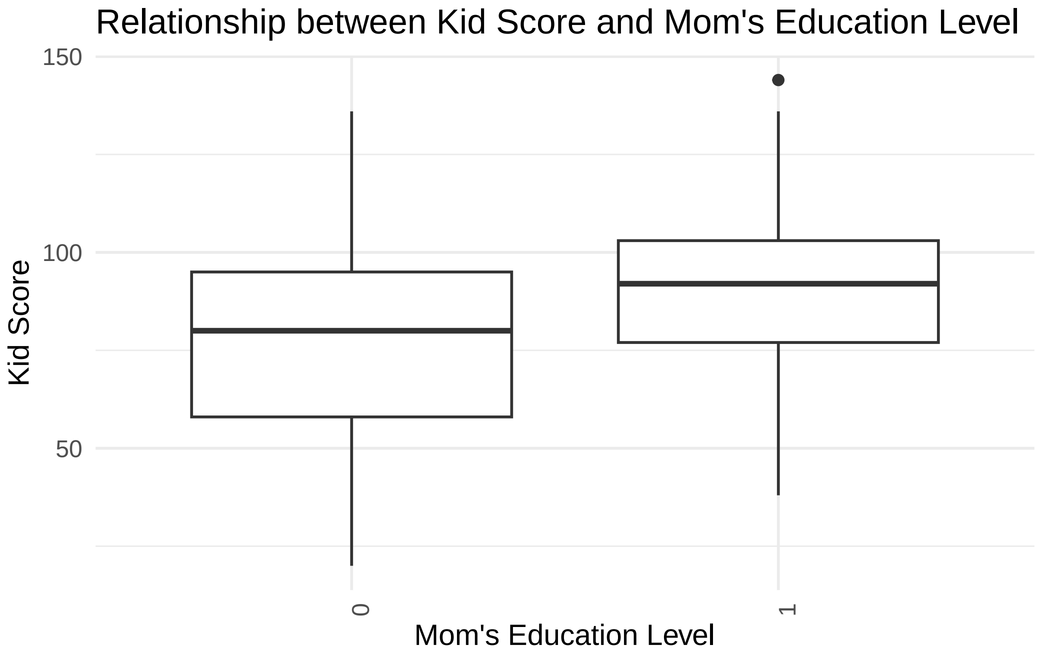
The scatter plot shows the relationship between children's IQ on the y-axis and their mothers' IQ on the x-axis. A line has been added to the plot, indicating the trend of the relationship. The regression line indicates a positive correlation, meaning as the Mom's IQ increases, there is a tendency for the kids' IQ to increase as well. However, the spread of the data points around the line suggests some variance, but we still can see the general trend of that.

```
kidiq |>
  ggplot(aes(x = factor(mom_age), y = kid_score)) +
  geom_boxplot() +
  labs(title = "Relationship between Kid Score and Mom's Age",
       x = "Mom's Age",
       y = "Kid Score") +
  theme_minimal()
```



The plot is a box plot that illustrates the relationship between children's IQ and their mothers' ages. The distribution of scores across different ages appears relatively consistent, with medians generally around 100 with some variation. For example, the box for age 29 is notably higher and shorter, suggesting that children of 29-year-old mothers have higher median scores and less variability in dataset. Moreover, outliers are present for several ages.

```
kidiq |>
  ggplot(aes(x = factor(mom_hs), y = kid_score)) +
  geom_boxplot() +
  labs(title = "Relationship between Kid Score and Mom's Education Level",
       x = "Mom's Education Level",
       y = "Kid Score") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```



Estimating mean, no covariates

In class we were trying to estimate the mean and standard deviation of the kid's test scores. The `kids2.stan` file contains a Stan model to do this. If you look at it, you will notice the first `data` chunk lists some inputs that we have to define: the outcome variable `y`, number of observations `N`, and the mean and standard deviation of the prior on `mu`. Let's define all these values in a `data` list.

```
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:

```r
fit <- stan(file = here("kids2.stan"),
            data = data,
            chains = 3,
            iter = 500)
```

```
Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
using C compiler: ‘Apple clang version 14.0.0 (clang-1400.0.29.202)’
using SDK: ‘MacOSX13.1.sdk’
clang -arch x86_64 -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-x86_64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/R
In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/RcppEigen/include/Eigen
namespace Eigen {
^
/Library/Frameworks/R.framework/Versions/4.3-x86_64/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-x86_64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.3-x86_64/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 2.8e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.28 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: 0.015 seconds (Warm-up)
Chain 1:                0.005 seconds (Sampling)
Chain 1:                0.02 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 / 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: 0.008 seconds (Warm-up)
Chain 2:                0.007 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 7e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.07 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: 0.015 seconds (Warm-up)
Chain 3:                0.005 seconds (Sampling)
Chain 3:                0.02 seconds (Total)
Chain 3:
```

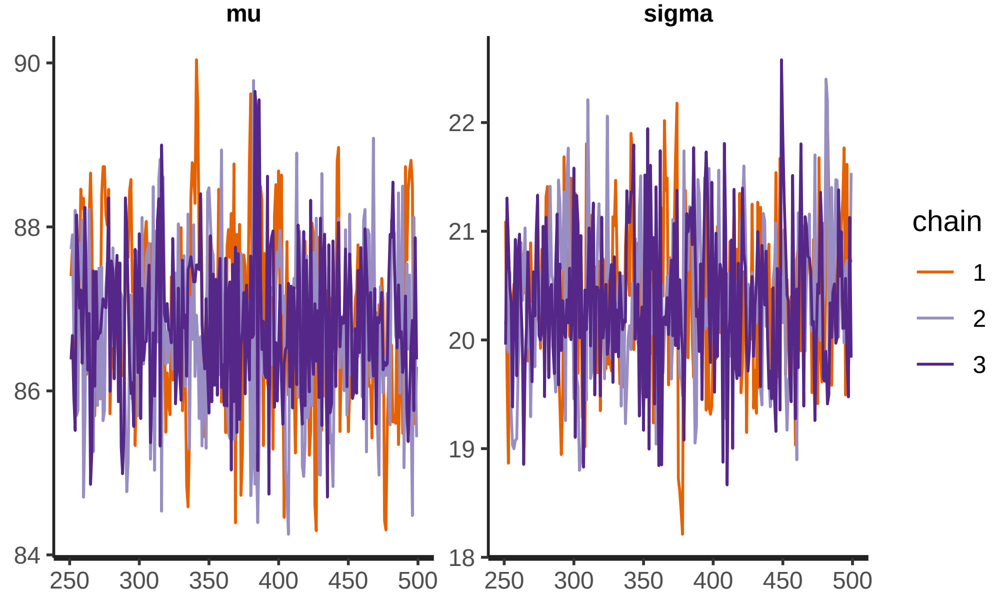
Look at the summary

```
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      86.82    0.05 1.01    84.84    86.1    86.82    87.52    88.73   477
sigma   20.36    0.03 0.69    19.02    19.9    20.35    20.79    21.74   533
lp__  -1525.80    0.06 1.02 -1528.49 -1526.2 -1525.49 -1525.05 -1524.79   253
      Rhat
mu    1.01
sigma 1.00
lp__  1.00

Samples were drawn using NUTS(diag_e) at Fri Feb 16 01:11:28 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).
```

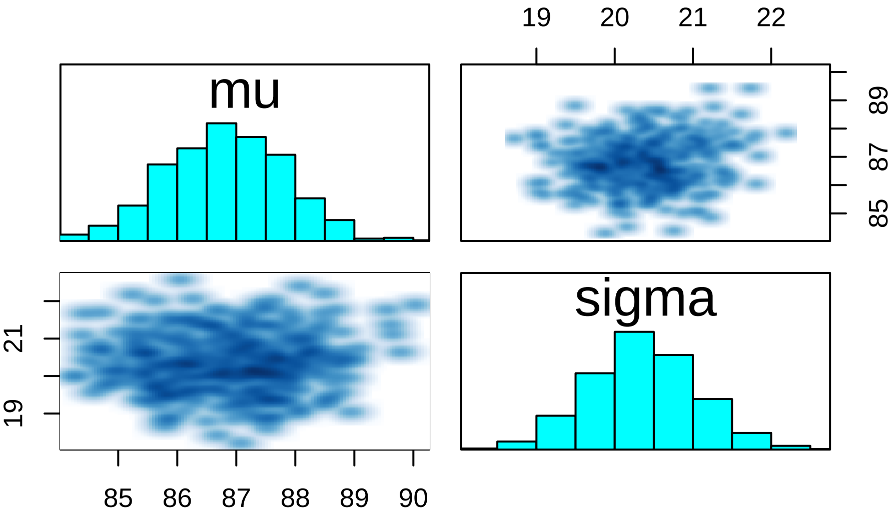
Traceplot

```
traceplot(fit)
```

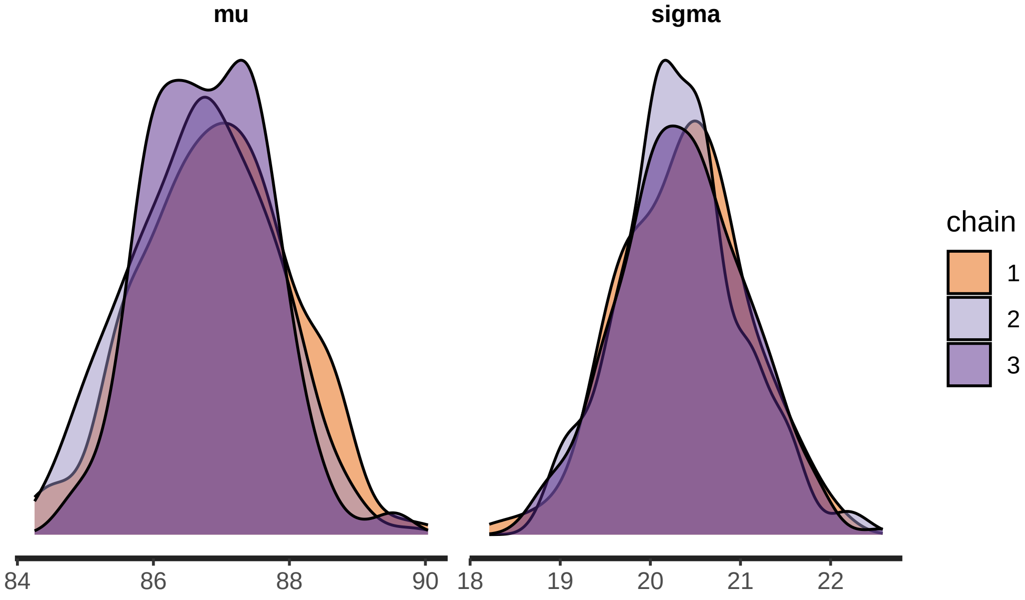


All looks fine.

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



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



Understanding output

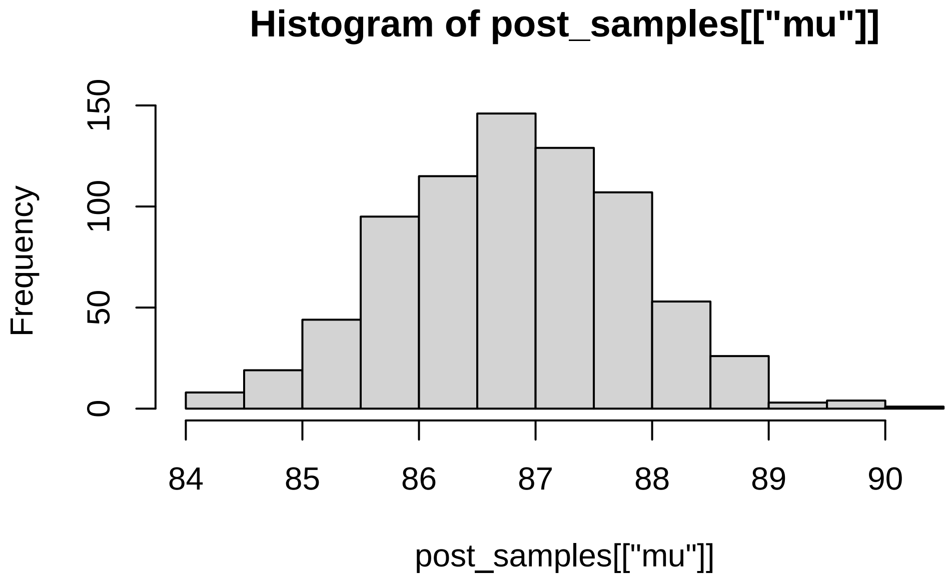
What does the model actually give us? A number of samples from the posteriors. To see this, we can use `extract` to get the samples.

```
post_samples <- extract(fit)
head(post_samples[["mu"]])
```

```
[1] 87.40021 87.91311 86.43134 86.31988 85.86394 88.03063
```

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] 86.82107
```

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

```
    2.5% 
84.83659 
```

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

```
   97.5% 
88.73492 
```

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          87.4
 2      1          2     2 mu          87.7
 3      1          3     3 mu          87.5
 4      1          4     4 mu          86.4
 5      1          5     5 mu          87.3
 6      1          6     6 mu          86.6
 7      1          7     7 mu          87.3
 8      1          8     8 mu          88.5
 9      1          9     9 mu          88.2
10      1         10    10 mu          88.3
# i 1,490 more rows
```

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

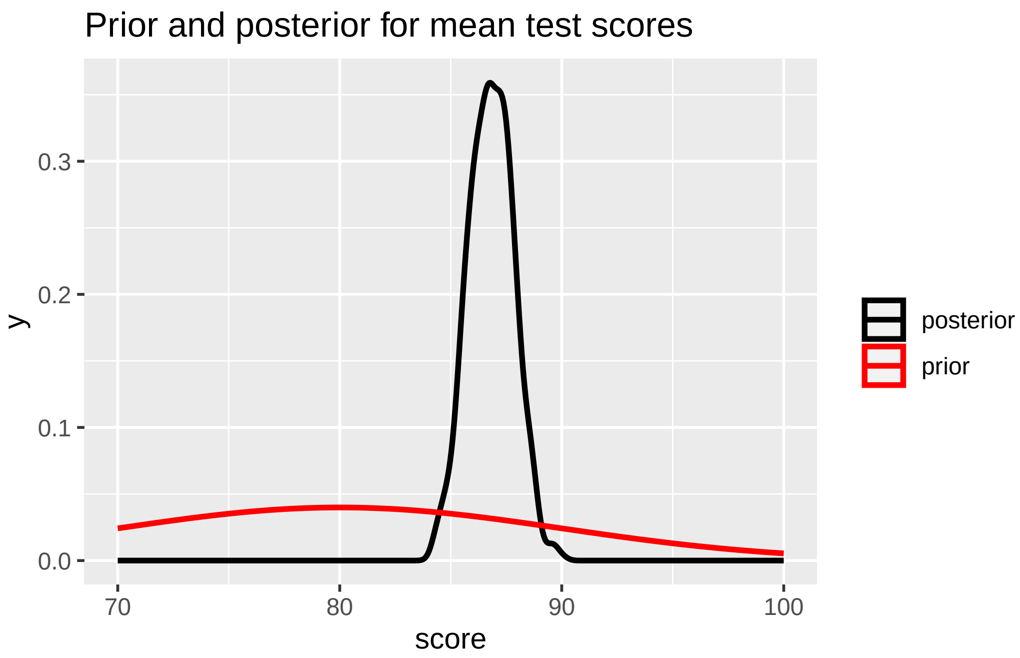
```
# A tibble: 750 x 5
   .chain .iteration .draw    mu sigma
    <int>      <int> <int> <dbl> <dbl>
 1      1          1     1  87.4  21.1
 2      1          2     2  87.7  19.7
 3      1          3     3  87.5  18.9
 4      1          4     4  86.4  20.2
 5      1          5     5  87.3  20.2
 6      1          6     6  86.6  20.3
 7      1          7     7  87.3  20.5
 8      1          8     8  88.5  20.4
 9      1          9     9  88.2  20.3
10      1         10    10  88.3  20.1
# 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          86.8   85.5   88.1    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")
```



Question 2

Change the prior to be much more informative (by changing the standard deviation to be 0.1). Rerun the model. Do the estimates change? Plot the prior and posterior densities.

```
y <- kidiq$kid_score
mu0 <- 80
sigma0 <- 0.1
# named list to input for stan function
data1 <- list(y = y,
            N = length(y),
            mu0 = mu0,
            sigma0 = sigma0)
```

```
fit1 <- stan(file = "kids2.stan",
              data = data1,
              chains = 3,
                iter = 500)
```

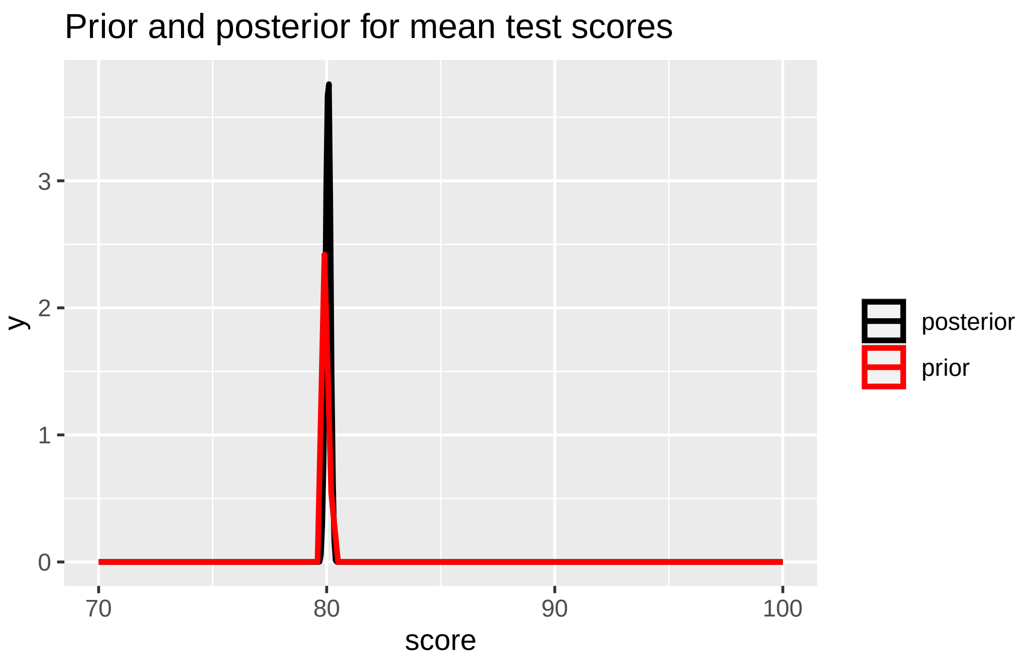
```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: 
Chain 1: Gradient evaluation took 7e-06 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.07 seconds.
Chain 1: Adjust your expectations accordingly!
Chain 1: 
Chain 1: 
Chain 1: Iteration:   1 / 500 [  0%]  (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: 0.006 seconds (Warm-up)
Chain 1:                0.005 seconds (Sampling)
Chain 1:                0.011 seconds (Total)
Chain 1: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2: 
Chain 2: Gradient evaluation took 5e-06 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.05 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: 0.006 seconds (Warm-up)
Chain 2:                0.006 seconds (Sampling)
Chain 2:                0.012 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 6e-06 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.06 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: 0.006 seconds (Warm-up)
Chain 3:                0.005 seconds (Sampling)
Chain 3:                0.011 seconds (Total)
Chain 3:
```

```r
print(fit1)
```

```
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      80.06    0.00 0.10    79.86    79.99    80.07    80.13    80.26   577
sigma   21.43    0.02 0.69    20.09    20.95    21.41    21.91    22.77   924
lp__ -1548.34    0.05 0.92 -1550.60 -1548.73 -1548.02 -1547.67 -1547.41   387
      Rhat
mu    1.01
sigma 1.00
lp__  1.01

Samples were drawn using NUTS(diag_e) at Fri Feb 16 01:11:31 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).
```

```r
dsamples <- fit1 |>
  gather_draws(mu, sigma)

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")
```



By changing the standard deviation to be 0.1, the estimates changed. $\hat{\mu}$ decreases from 86.71446 to 80.06 and $\hat{\sigma}$ changes from 20.33741 to 21.41.

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

$$Score = \alpha + \beta X$$

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 = here("kids3.stan"),
            data = data,
            iter = 1000)
```

```
Running /Library/Frameworks/R.framework/Resources/bin/R CMD SHLIB foo.c
using C compiler: ‘Apple clang version 14.0.0 (clang-1400.0.29.202)’
using SDK: ‘MacOSX13.1.sdk’
clang -arch x86_64 -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-x86_64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/R
In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/RcppEigen/include/Eigen
namespace Eigen {
^
/Library/Frameworks/R.framework/Versions/4.3-x86_64/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-x86_64/Resources/library/S
In file included from /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library/R
/Library/Frameworks/R.framework/Versions/4.3-x86_64/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 0.000148 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 1.48 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: 0.197 seconds (Warm-up)
Chain 1:                0.106 seconds (Sampling)
Chain 1:                0.303 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 2.6e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.26 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: 0.214 seconds (Warm-up)
Chain 2:                0.102 seconds (Sampling)
Chain 2:                0.316 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 2.5e-05 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.25 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: 0.198 seconds (Warm-up)
Chain 3:                0.088 seconds (Sampling)
Chain 3:                0.286 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 2.4e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.24 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: 0.144 seconds (Warm-up)
Chain 4:                0.113 seconds (Sampling)
Chain 4:                0.257 seconds (Total)
Chain 4: 
```

Question 3

a) Confirm that the estimates of the intercept and slope are comparable to results from `lm()`
b) Do a `pairs` plot to investigate the joint sample distributions of the slope and intercept. Comment briefly on what you see. Is this potentially a problem?

```
print(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      77.93    0.08 2.15    73.79    76.46    77.95    79.38    82.08
beta[1]    11.24    0.09 2.41     6.40     9.61    11.23    12.90    16.00
sigma      19.84    0.02 0.64    18.59    19.40    19.84    20.28    21.11
lp__    -1514.44    0.05 1.27 -1517.60 -1515.10 -1514.09 -1513.50 -1512.96
        n_eff Rhat
alpha     814    1
beta[1]   786    1
sigma     965    1
lp__      663    1
```

```
Samples were drawn using NUTS(diag_e) at Fri Feb 16 01:12:37 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).
```

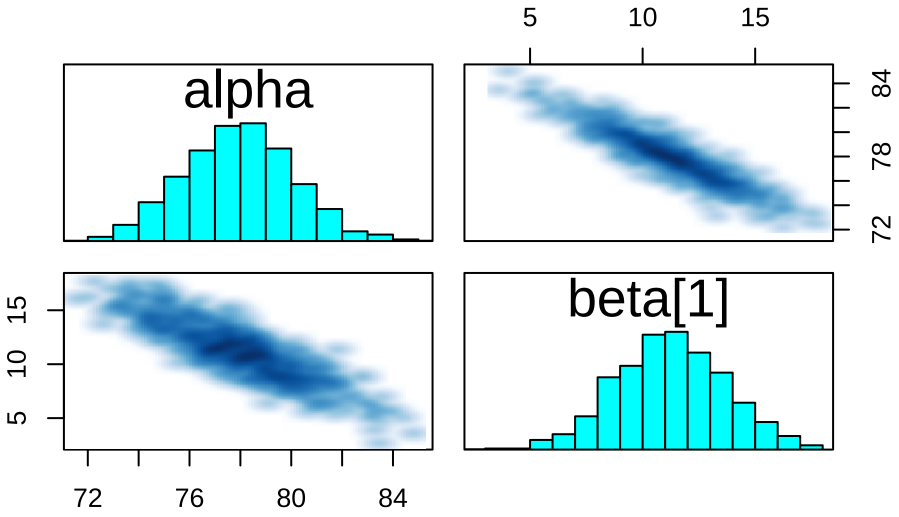
```r
model <- lm(kid_score ~ mom_hs, data=kidiq)
summary(model)
```

```

Call:
lm(formula = kid_score ~ mom_hs, data = kidiq)

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 ***
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
```

The estimate for the intercept from lm() is 77.548, and for the bayes regression, the intercept is about 78, 2 values are closed. Meanwhile, the estimate for slope from lm() is 11.771, which is very close to slope inthe bayes regression, which is about 11.30. Thus, the estimates of the intercept and slope are comparable to results from lm().

b) Do a `pairs` plot to investigate the joint sample distributions of the slope and intercept. Comment briefly on what you see. Is this potentially a problem?

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

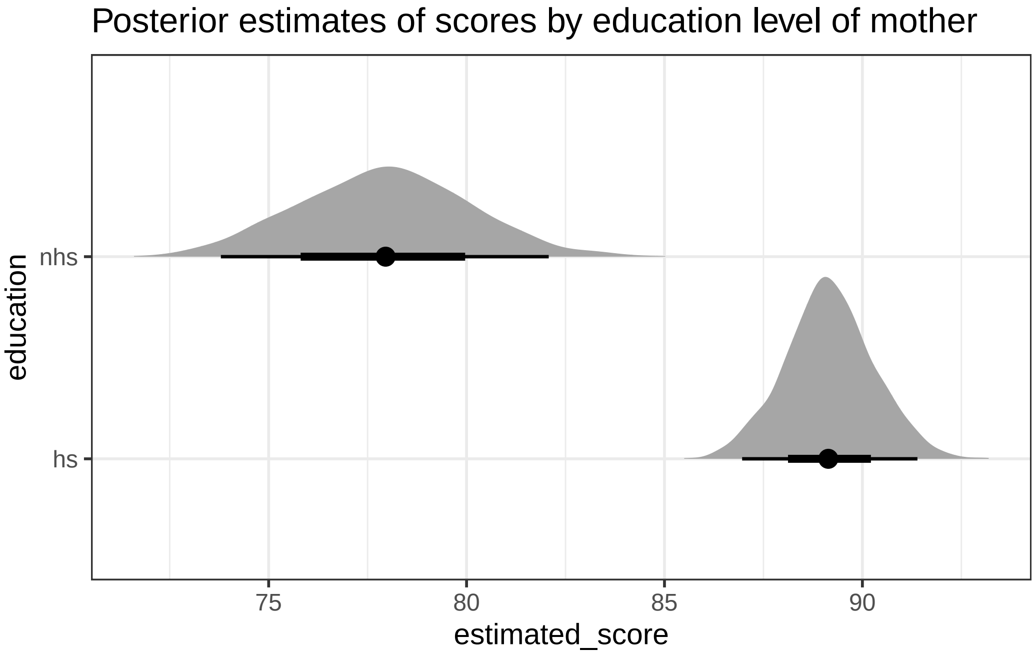


From the `pairs` plot, we can find there is a negative linear relationship between the intercept (alpha) and the slope (beta1). Multicollinearity can lead to instability in the estimation of the model parameters, resulting in unreliable and highly variable estimates that are sensitive to minor changes in the model or the data.

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,
          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")
```



Question 4

Add in mother's IQ as a covariate and rerun the model. Please mean center the covariate before putting it into the model. Interpret the coefficient on the (centered) mum's IQ.

```
kidiq$centered_mom_iq <- kidiq$mom_iq - mean(kidiq$mom_iq)
X <- as.matrix(kidiq[, c("mom_hs", "centered_mom_iq")])
K <- 2
data3 <- list(y = y,
            N = length(y),
            X = X,
            K = K
)

fit4 <- stan(file = "kids3.stan",
             data = data3,
             iter = 1000)
```

```
SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 1).
Chain 1: 
Chain 1: Gradient evaluation took 2.7e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.27 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: 0.182 seconds (Warm-up)
Chain 1:                0.128 seconds (Sampling)
Chain 1:                0.31 seconds (Total)
Chain 1: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 2).
Chain 2: 
Chain 2: Gradient evaluation took 2.8e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.28 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: 0.187 seconds (Warm-up)
Chain 2:                0.136 seconds (Sampling)
Chain 2:                0.323 seconds (Total)
Chain 2: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 3).
Chain 3: 
Chain 3: Gradient evaluation took 2.5e-05 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.25 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: 0.178 seconds (Warm-up)
Chain 3:                0.133 seconds (Sampling)
Chain 3:                0.311 seconds (Total)
Chain 3: 

SAMPLING FOR MODEL 'anon_model' NOW (CHAIN 4).
Chain 4: 
Chain 4: Gradient evaluation took 2.5e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.25 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: 0.222 seconds (Warm-up)
Chain 4:                0.13 seconds (Sampling)
Chain 4:                0.352 seconds (Total)
Chain 4: 
```

```
print(fit4)
```

```
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      82.45    0.05 1.95    78.51    81.19    82.45    83.77    86.21
beta[1]     5.50    0.06 2.21     1.48     3.98     5.48     6.94    10.15
beta[2]     0.57    0.00 0.06     0.46     0.53     0.57     0.61     0.68
sigma      18.10    0.02 0.63    16.91    17.65    18.08    18.50    19.32
lp__    -1474.42    0.05 1.45 -1478.07 -1475.08 -1474.06 -1473.38 -1472.67
        n_eff Rhat
alpha    1388 1.00
beta[1]  1356 1.00
beta[2]  1615 1.00
sigma    1362 1.00
lp__      853 1.01

Samples were drawn using NUTS(diag_e) at Fri Feb 16 01:12:40 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).
```

The interpretation of coefficient on the (centered) mum's IQ is that keep other covariates unchanged, on average, an unit increases in the mom's IQ, the kid's score is expected to increase by 0.57.

Question 5

Confirm the results from Stan agree with `lm()`

```r
model1 <- lm(kid_score ~ mom_hs + centered_mom_iq, data=kidiq)
summary(model1)
```

```

Call:
lm(formula = kid_score ~ mom_hs + centered_mom_iq, data = kidiq)

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 ***
mom_hs           5.95012    2.21181   2.690  0.00742 ** 
centered_mom_iq  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
```

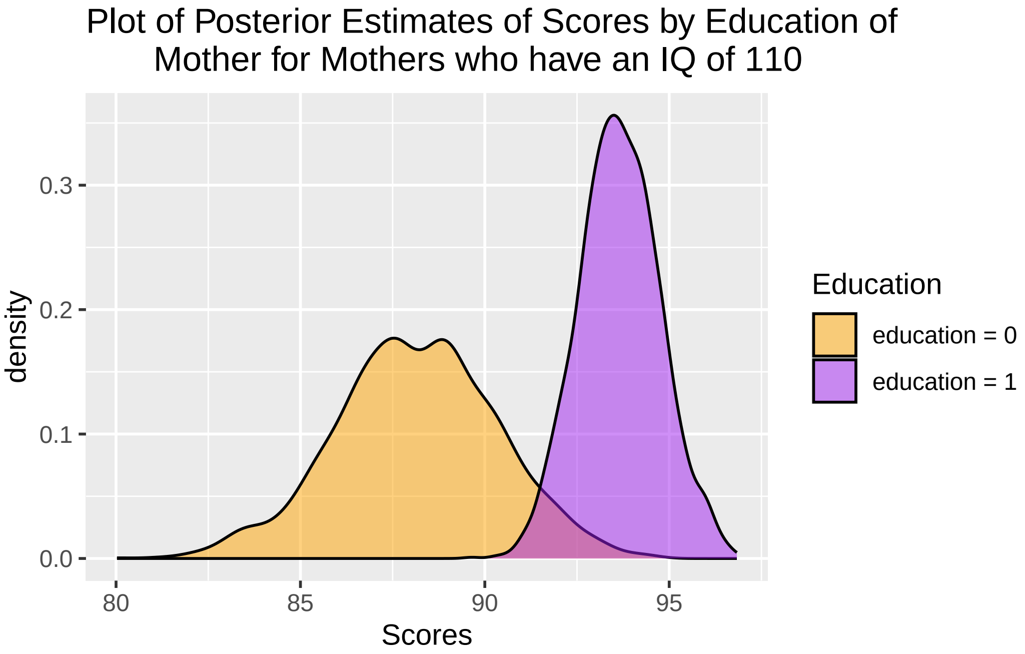
From the lm() result, the coefficient on the (centered) mum's IQ is 0.56391, which is very close to the results from Stan (0.57). Thus, results from Stan agree with `lm()`

Question 6

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

```r
ext_fit <- extract(fit4)
alpha_post <- ext_fit$alpha
beta_post <- ext_fit$beta
sigma<-ext_fit$sigma
b1<-beta_post[,1]
b2<-beta_post[,2]
```

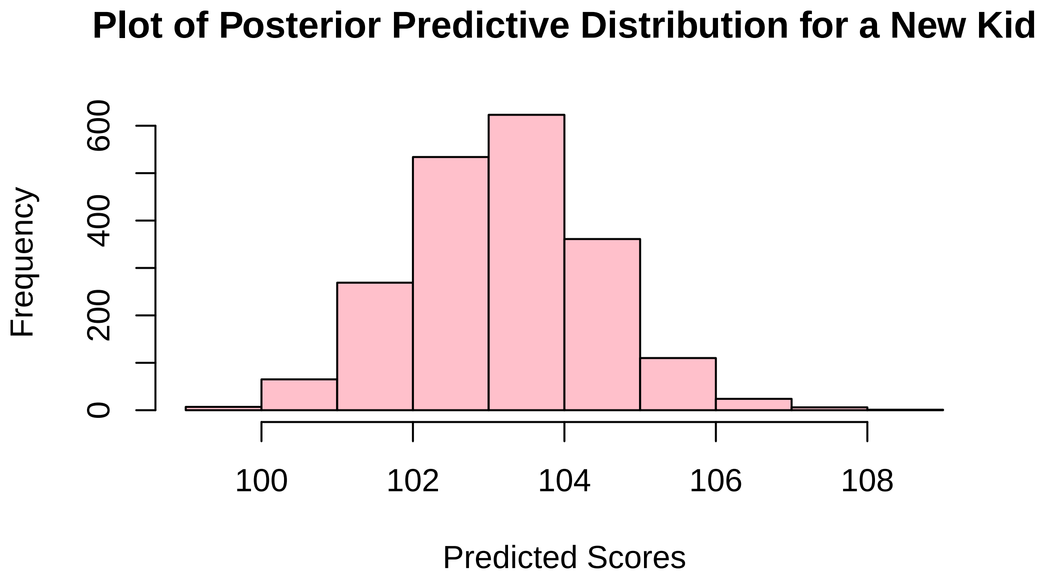
```
posterior0 <- alpha_post  +  b1 * 0 + b2  * (110 - mean(kidiq$mom_iq))
posterior1 <- alpha_post  + b1 * 1 + b2 * (110 - mean(kidiq$mom_iq))
df <- data.frame(
  Scores = c(posterior0, posterior1),
  Education = rep(c("education = 0", "education = 1"), each = length(posterior0))
)
ggplot(df, aes(x = Scores, fill = Education)) +
  geom_density(alpha = 0.5) +
  labs(title = "Plot of Posterior Estimates of Scores by Education of
       Mother for Mothers who have an IQ of 110") +
  scale_fill_manual(values = c("orange", "purple"))
```



Question 7

Generate and plot (as a histogram) samples from the posterior predictive distribution for a new kid with a mother who graduated high school and has an IQ of 95.

```
posterior3 <- alpha_post  + b1 + b2 * (95 - mean(kidiq$mom_iq)) + sigma

hist(posterior3, main = "Plot of Posterior Predictive Distribution for a New Kid",
     xlab = "Predicted Scores",col = "pink")
```


Plot of Posterior Predictive Distribution for a New Kid
Frequency
0
200
400
600
100
102
104
106
108
Predicted Scores