Stat 136 (Bayesian Statistics)

Lesson 4.1 Bayesian Regression

Introduction

Regression is a class of statistical techniques to understand the relationship between an outcome variable (also called a criterion/response/dependent variable) and one or more predictor variables (also called explanatory/independent variables).

It is assumed that everyone is familiar with linear regression analysis and how to fit and interpret linear regression models in R.

In this lesson, we shall learn how to perform Bayesian regression using the **rstanarm** package. For more information about the package please refer to this page **https://cran.r-project.org/web/packages/rstanarm/rstanarm.pdf**.

We are using the **rstanarm** package because it is very flexible and it is easy to specify prior densities.

We will use a data set, *kidiq*, that is available in the **rstanarm** package. A brief description of the data is given below.

```
kidiq

    Data from a survey of adult American women and their children (a subsample
    from the National Longitudinal Survey of Youth).

    Source: Gelman and Hill (2007)

    434 obs. of 5 variables

        kid_score Child's IQ score
        mom_hs Indicator for whether the mother has a high school degree
        mom_iq Mother's IQ score
        mom_work 1 = did not work in first three years of child's life
                 2 = worked in 2nd or 3rd year of child's life
                 3 = worked part-time in first year of child's life
                 4 = worked full-time in first year of child's life
        mom_age Mother's age
```

We begin by loading the required packages and reading the data into R.

```r
library(tidyverse)
library(rstanarm)
library(bayesplot)
library(brms)
library(foreign)

kidiq <-read.csv("kidiq.csv")
head(kidiq)
```

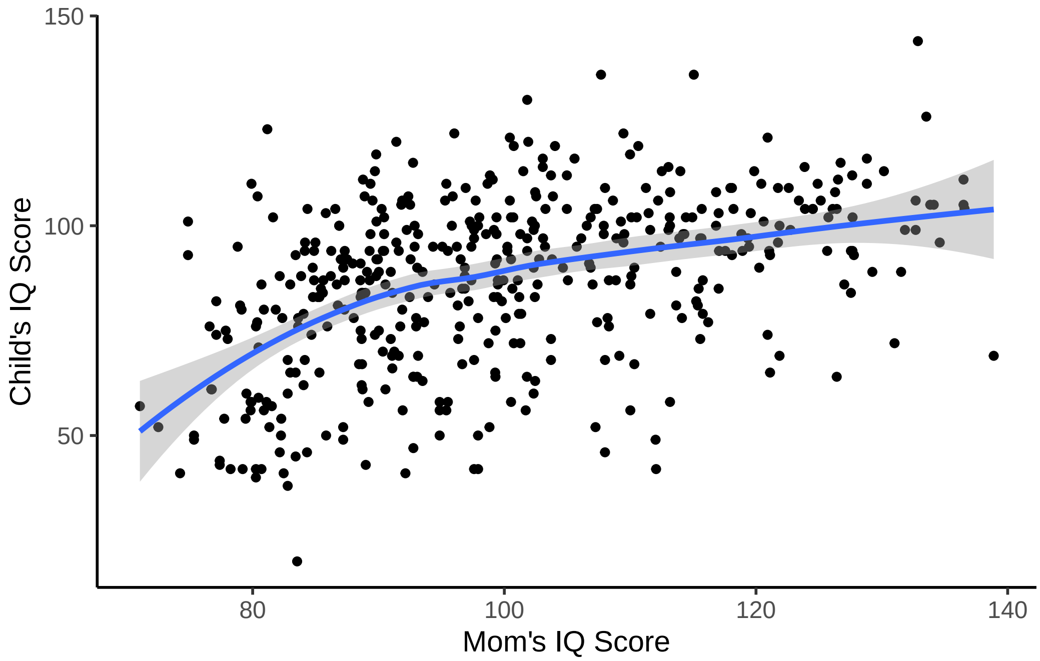
```
  X kid_score mom_hs    mom_iq mom_work mom_age
1 1        65      1 121.11753        4      27
2 2        98      1  89.36188        4      25
3 3        85      1 115.44316        4      27
4 4        83      1  99.44964        3      25
5 5       115      1  92.74571        4      27
6 6        98      0 107.90184        1      18
```

```r
psych::describe(kidiq) #Generates summary statistics of the variables in the data
```

```
          vars   n   mean     sd median trimmed    mad   min    max  range
X            1 434 217.50 125.43 217.50  217.50 160.86  1.00 434.00 433.00
kid_score    2 434  86.80  20.41  90.00   87.93  19.27 20.00 144.00 124.00
mom_hs       3 434   0.79   0.41   1.00    0.86   0.00  0.00   1.00   1.00
mom_iq       4 434 100.00  15.00  97.92   99.11  15.89 71.04 138.89  67.86
mom_work     5 434   2.90   1.18   3.00    2.99   1.48  1.00   4.00   3.00
mom_age      6 434  22.79   2.70  23.00   22.71   2.97 17.00  29.00  12.00
           skew kurtosis   se
X          0.00    -1.21 6.02
kid_score -0.46    -0.19 0.98
mom_hs    -1.39    -0.07 0.02
mom_iq     0.47    -0.59 0.72
mom_work  -0.45    -1.39 0.06
mom_age    0.18    -0.65 0.13
```

We will first use mother's score on an IQ test to predict the child's test score, as shown in the following scatter plot.

```r
kidiq %>%
  ggplot(aes(x = mom_iq, y = kid_score)) +
  geom_point(size = 1.1) +
  geom_smooth()+
  labs(x = "Mom's IQ Score",
       y = "Child's IQ Score") +
  theme_classic()
```



Suppose we fit the following simple linear regression model.

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

where: y_i is the IQ score of the i^{th} kid; x_i is the IQ score of the mom of the i^{th} kid.

As usual β_0 is the regression intercept which is the kids' mean IQ score when mom's IQ score is zero; and β is the called regression slope or regression coefficient which is equal to the mean change in kid's IQ score if mom's IQ score increases by 1 unit; and $\epsilon_i \sim N(0, \sigma^2)$ is the random error component which accounts for the variation in kid's IQ score which cannot be explained by x_i. It is assumed that the variability (σ) of kid's IQ scores is constant across mom's IQ scores.

Estimation via rstanarm package

We start with a Bayesian model with uninformative prior.

```
model1 <- stan_glm(kid_score ~  mom_iq,
                 data = kidiq)
```

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

```
Chain 1:
Chain 1: Gradient evaluation took 4.9e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.49 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: 0.038 seconds (Warm-up)
Chain 1:                0.05 seconds (Sampling)
Chain 1:                0.088 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 1.9e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.19 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: 0.037 seconds (Warm-up)
Chain 2:                0.062 seconds (Sampling)
Chain 2:                0.099 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'continuous' 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 / 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: 0.03 seconds (Warm-up)
Chain 3:                0.046 seconds (Sampling)
Chain 3:                0.076 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 1.8e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.18 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: 0.032 seconds (Warm-up)
Chain 4:                0.046 seconds (Sampling)
Chain 4:                0.078 seconds (Total)
Chain 4:
```

```r
summary(model1,digits=5)
```

```
Model Info:
 function:     stan_glm
 family:       gaussian [identity]
 formula:      kid_score ~ mom_iq
 algorithm:    sampling
 sample:       4000 (posterior sample size)
 priors:       see help('prior_summary')
 observations: 434
 predictors:   2

Estimates:
              mean     sd       10%      50%      90%
(Intercept) 25.79516  6.07303 18.04265 25.68084 33.76512
mom_iq       0.61002  0.05992  0.53048  0.61096  0.68562
sigma       18.31052  0.62200 17.53347 18.29930 19.12179

Fit Diagnostics:
           mean     sd       10%      50%      90%
mean_PPD 86.80503  1.24421 85.20673 86.81577 88.41267

The mean_ppd is the sample average posterior predictive distribution of the outcome variable

MCMC diagnostics
              mcse    Rhat    n_eff
(Intercept)   0.09806 0.99991 3836
mom_iq        0.00097 0.99989 3807
```

```
sigma         0.01019 1.00065 3723
mean_PPD      0.02114 1.00048 3465
log-posterior 0.02782 1.00052 1952

For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective
```

We can generate credible intervals of the estimates as follows. These credible intervals can be used also to test significance of the estimates.

```r
posterior_interval(model1, prob=0.95)
```

```
                  2.5%      97.5%
(Intercept) 14.1256899 37.7336201
mom_iq       0.4925145  0.7248289
sigma       17.1473089 19.5836634
```

Since the 95% credible intervals for the three parameters do not include zero, indicating the parameter estimates are significant;y different from zero.

Before we proceed to other inferences let us take a look at the prior distributions that were used in the above model fitting. The default priors in **rstanarm** are intended to be weakly informative and, in general, unless a lot of prior information is available, we recommend weakly informative priors for the parameters of a regression model. A weakly informative prior that reflects the expected magnitude of the parameters based on the scales of the variables will not strongly impact the posterior, but will provide regularization to stabilize computation and avoid overfitting, while still allowing for extreme values when warranted by the data (Gelman, Jakulin, Pittau, & Su, 2008; Stan Development Team, 2017).

```r
prior_summary(model1)
```

```
Priors for model 'model1'
------
Intercept (after predictors centered)
  Specified prior:
    ~ normal(location = 87, scale = 2.5)
  Adjusted prior:
    ~ normal(location = 87, scale = 51)

Coefficients
  Specified prior:
    ~ normal(location = 0, scale = 2.5)
```

```
  Adjusted prior:
    ~ normal(location = 0, scale = 3.4)

Auxiliary (sigma)
  Specified prior:
    ~ exponential(rate = 1)
  Adjusted prior:
    ~ exponential(rate = 0.049)
------
See help('prior_summary.stanreg') for more details
```

The *stan_glm()* also allows the user to specify his/her own prior distributions for the model parameters. This is illustrated in the following code chunk.

```
model2 <- stan_glm(kid_score ~  mom_iq,
                   prior = normal(0,3),
                   prior_intercept = normal(0,3),
                   data = kidiq)
```

```
SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 2e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.2 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: 0.039 seconds (Warm-up)
Chain 1:                0.046 seconds (Sampling)
```

```
Chain 1:                0.085 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'continuous' 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 / 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: 0.03 seconds (Warm-up)
Chain 2:                0.061 seconds (Sampling)
Chain 2:                0.091 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 1e-05 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.1 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: 0.036 seconds (Warm-up)
Chain 3:                0.056 seconds (Sampling)
Chain 3:                0.092 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'continuous' 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 / 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: 0.031 seconds (Warm-up)
Chain 4:                0.055 seconds (Sampling)
Chain 4:                0.086 seconds (Total)
Chain 4:
```

```r
summary(model2,digits=5)
```

```
Model Info:
 function:     stan_glm
 family:       gaussian [identity]
 formula:      kid_score ~ mom_iq
```

```
 algorithm:    sampling
 sample:       4000 (posterior sample size)
 priors:       see help('prior_summary')
 observations: 434
 predictors:   2

Estimates:
              mean     sd       10%      50%      90%   
(Intercept) 17.68548  6.51457  9.25359 17.83075 25.93762
mom_iq       0.61024  0.06433  0.52986  0.60908  0.69410
sigma       20.04580  0.81804 19.01189 20.03765 21.11619

Fit Diagnostics:
           mean     sd       10%      50%      90%   
mean_PPD 78.72840  1.46306 76.82761 78.75784 80.60330

The mean_ppd is the sample average posterior predictive distribution of the outcome variable

MCMC diagnostics
              mcse    Rhat    n_eff
(Intercept)   0.11304 0.99927 3321 
mom_iq        0.00111 0.99927 3349 
sigma         0.01679 1.00010 2373 
mean_PPD      0.02712 0.99967 2911 
log-posterior 0.02945 1.00157 1689 

For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective
```

```r
posterior_interval(model2, prob=0.95)
```

```
                  2.5%      97.5%
(Intercept)  4.4428809 30.1045937
mom_iq       0.4868192  0.7403286
sigma       18.4819729 21.6500639
```

Checking Sampling Quality

After estimation, researchers should look for signs that the chains might not have converged and check that there is a large enough effective sample size for the analysis. The *summary()* function provides both a summary of parameter estimates as well as diagnostic information about the sampling quality. The output consists of a short description of the analysis that was carried out followed by a summary of the parameter estimates (interpreted in Step 4) and the logposterior, and finally three quantities related to the performance of the sampler: Monte Carlo standard error (MCSE), $\hat{R}$ (Rhat) and effective sample size (n_eff).

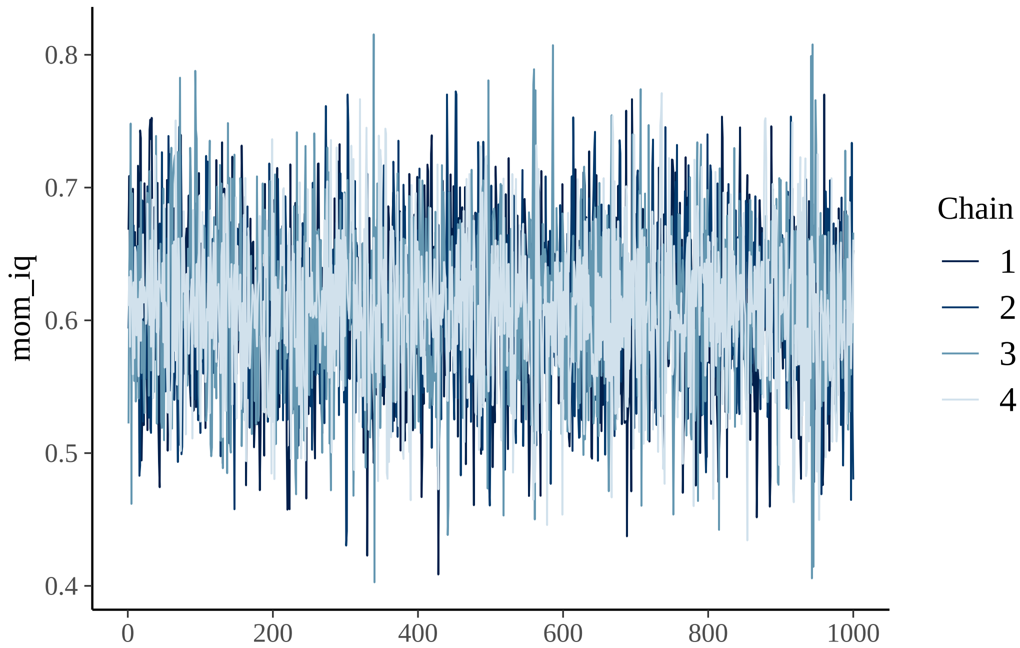
Monte Carlo Standard Error (MCSE) quantifies the uncertainty or inaccuracy of estimates obtained from a Monte Carlo simulation. It essentially measures how much the estimate might vary if the simulation were run multiple times with different random seeds. A smaller MCSE indicates a more precise and reliable estimate.

R-hat provides a way to quantify whether multiple chains of a model are converging to the same distribution. In ideal circumstances, R-hat should approach 1, indicating that all chains are sampling from the same underlying distribution. Values significantly greater than 1.1 suggest that the chains have not yet converged, and further iterations or changes to the model are needed.

The effective sample size (n_eff) is a measure of how many independent samples you effectively have from your MCMC chain, considering the autocorrelation between successive samples. It helps you understand how well your MCMC chain is exploring the posterior distribution and whether you have sufficient data to make reliable inferences. A common rule of thumb is to aim for an n_eff that is at least 10% of your total sample size (e.g., if you have 10,000 samples, you want n_eff to be at least 1000).

We can also check visually if the chains converge to the same distribution.

```
plot(model1, "trace",
     pars = "mom_iq")
```



Posterior predictive checking

To visually assess the fit of the model to the data, we can compare the observed data to datasets simulated according to our assumed data generating process and the posterior draws of the model parameters. The code below uses the *pp_check()* function to plot a smoothed kernel density estimate of the original data, overlaying the density estimates from 100 generated data sets from the posterior predictive distribution:

```
library(patchwork)
g1 <- pp_check(model1, nreps = 100) +
  labs(x = "Mom's IQ", title = "Model 1")

g2 <- pp_check(model2, nreps = 100) +
  labs(x = "Mom's IQ", title = "Model 2")

g1/g2
```

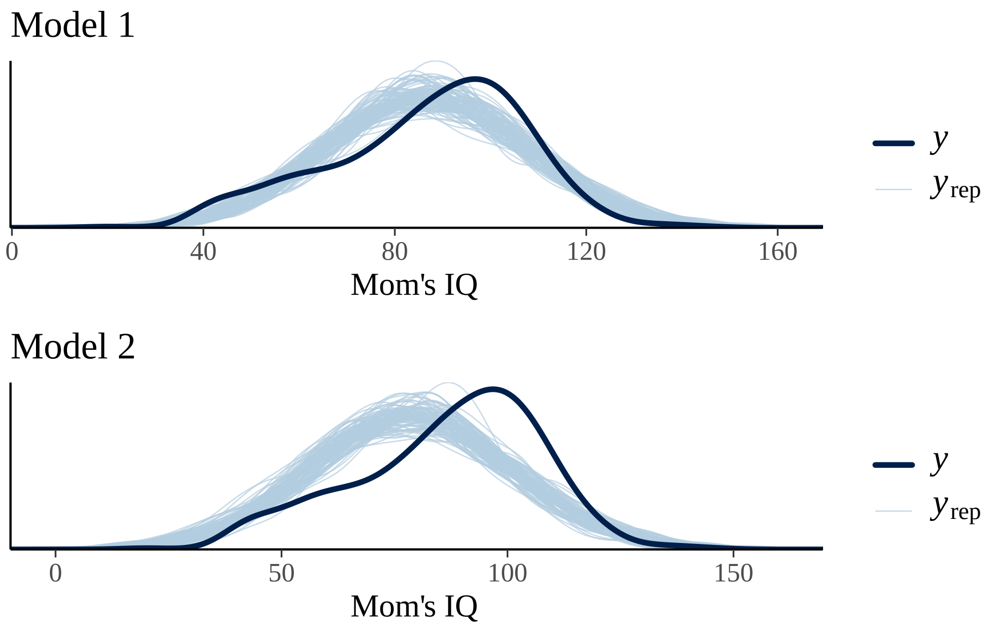


Figure 1: Posterior Predictive Distributions

For models that fit the data well, this type of plot will show that the draws from the posterior predictive distribution (thin light blue lines) and the observed data (thick dark blue line) have similar distributions.

In the above plots, Model 1 seems to fit the data better than Model 2.

An alternative plot for predictive checking is given below.

```
g1 <- pp_check(model1,
         "stat",
         nreps = 100) +
  labs(x = "Mom's IQ", title = "Model 1")

g2 <- pp_check(model2,
               "stat",
               nreps = 100) +
  labs(x = "Mom's IQ", title = "Model 2")

g1/g2
```

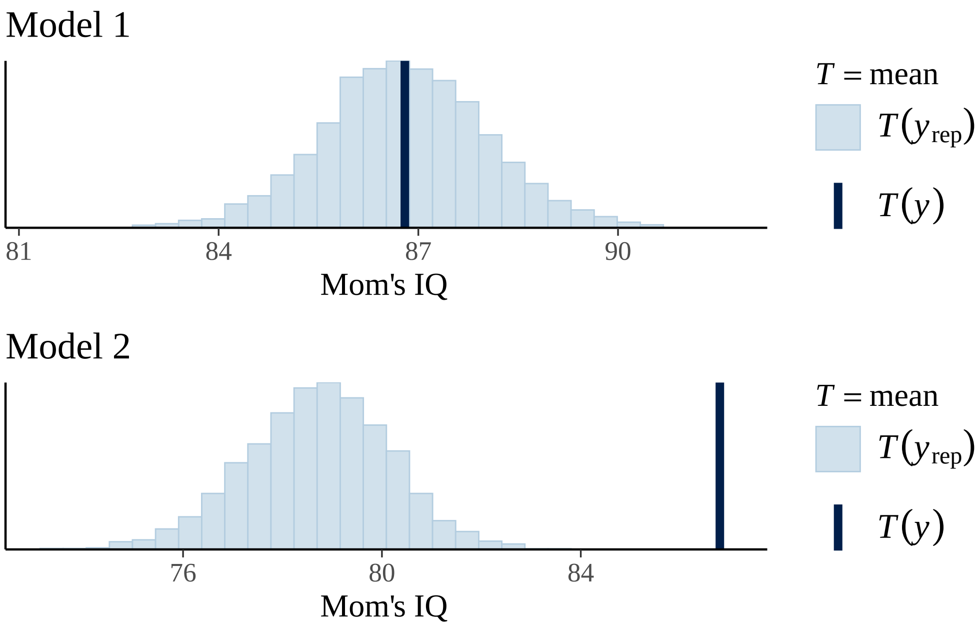


Figure 2: Posterior Predictive Distributions

In the above plots, it is confirmed that Model 1 fits the data better than Model 2.

The shinystan package

The user can easily explore the posterior further through the **shinystan** R package, which provides a graphical user interface for interactively exploring **rstanarm** models (or any other models fit using MCMC). Visual and numeric checks are both available from **shinystan** via a user-friendly graphical user interface. With **shinystan** researchers can look at both estimates and diagnostics, and it is easy to customize exportable tables and graphics. **shinystan** provides a wide variety of tools for visualizing and summarizing the posterior distribution and diagnosing MCMC problems.

```
launch_shinystan(model1)
```

Model comparison

The **loo** (leave-one-out cross-validation) package provides a more comprehensive approach to model comparison in a Bayesian setting. The use of leave-one-out cross-validation approach often results to a better measure of model fit than AIC. **loo**'s LOOIC (leave-one-out information criterion) and WAIC (widely applicable information criterion) are often preferred for Bayesian models, as they better account for uncertainty in the parameters.

```r
library(loo)
```

```
This is loo version 2.8.0

- Online documentation and vignettes at mc-stan.org/loo

- As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'c

- Windows 10 users: loo may be very slow if 'mc.cores' is set in your .Rprofile file (see ht
```

```r
lmodel1 <- loo(model1)
lmodel2 <- loo(model2)

data.frame("LOOIC Model 1" = lmodel1$estimates[3],
           "LOOIC Model 2" = lmodel2$estimates[3])
```

```
  LOOIC.Model.1 LOOIC.Model.2
1      3757.231      3834.855
```

```r
loo_compare(lmodel1, lmodel2)
```

```
       elpd_diff se_diff
model1   0.0       0.0
model2 -38.8       8.6
```

The above results confirm our initial impression that Model 1 fits the data better than Model 2. Using the leave-one-out cross-validation approach, Model 1 has lower LOOIC than Model 2.

Generate predicted outcomes

The posterior predictive distribution is the distribution of the outcome implied by the model after using the observed data to update our beliefs about the unknown parameters in the model. Simulating data from the posterior predictive distribution using the observed predictors is useful for checking the fit of the model. Drawing from the posterior predictive distribution at interesting values of the predictors also lets us visualize how a manipulation of a predictor affects (a function of) the outcome(s). With new observations of predictor variables we can use the posterior predictive distribution to generate predicted outcomes.

For example, we might be interested to know the expected IQ score of a child whose mother has IQ of 95. We can use the posterior distribution to predict this kid's IQ score.

```
nd <- data.frame(mom_iq = 95)

score.pred <- posterior_predict(model1, newdata = nd)

score.intvl <- predictive_interval(model1, newdata = nd, prob = 0.95)

print(c(mean(score.pred),score.intvl))
```

```
[1]  84.02957  47.74335 119.08722
```

Bayesian approach to multiple linear regression

Suppose this time we wish to fit the following statistical model to the data.

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \beta_4 x_{4i} + \epsilon_i$$

where y_i is the kid's IQ score; $x_{1i} = mom\_hs$; $x_{2i} = mom\_iq$; $x_{3i} = mom\_work$; $x_{4i} = mom\_age$.

Before we fit a MLR model, we first transform the variables mom_hs and mom_work as factors.

```
kidiq1 <- kidiq %>%
  mutate(mom_hs = as.factor(mom_hs),
         mom_work = as.factor(mom_work))
```

Next we fit the MLR model using default priors.

```r
model3 <- stan_glm(kid_score ~  mom_hs + mom_iq + mom_work + mom_age,
                  data = kidiq1)
```

```
SAMPLING FOR MODEL 'continuous' NOW (CHAIN 1).
Chain 1:
Chain 1: Gradient evaluation took 1.9e-05 seconds
Chain 1: 1000 transitions using 10 leapfrog steps per transition would take 0.19 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: 0.054 seconds (Warm-up)
Chain 1:                0.063 seconds (Sampling)
Chain 1:                0.117 seconds (Total)
Chain 1:

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 2).
Chain 2:
Chain 2: Gradient evaluation took 1.4e-05 seconds
Chain 2: 1000 transitions using 10 leapfrog steps per transition would take 0.14 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: 0.051 seconds (Warm-up)
Chain 2:                0.059 seconds (Sampling)
Chain 2:                0.11 seconds (Total)
Chain 2:

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 3).
Chain 3:
Chain 3: Gradient evaluation took 1e-05 seconds
Chain 3: 1000 transitions using 10 leapfrog steps per transition would take 0.1 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: 0.051 seconds (Warm-up)
Chain 3:                0.054 seconds (Sampling)
Chain 3:                0.105 seconds (Total)
Chain 3:

SAMPLING FOR MODEL 'continuous' NOW (CHAIN 4).
Chain 4:
Chain 4: Gradient evaluation took 1e-05 seconds
Chain 4: 1000 transitions using 10 leapfrog steps per transition would take 0.1 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: 0.047 seconds (Warm-up)
Chain 4:                0.069 seconds (Sampling)
Chain 4:                0.116 seconds (Total)
Chain 4:
```

```
summary(model3,digits=5)
```

```
Model Info:
 function:     stan_glm
 family:       gaussian [identity]
 formula:      kid_score ~ mom_hs + mom_iq + mom_work + mom_age
 algorithm:    sampling
 sample:       4000 (posterior sample size)
 priors:       see help('prior_summary')
 observations: 434
 predictors:   7

Estimates:
              mean     sd       10%      50%      90%
(Intercept) 20.05972  9.34544  7.86889 20.16699 31.88343
mom_hs1      5.43049  2.29578  2.53770  5.38577  8.35882
mom_iq       0.55401  0.06197  0.47473  0.55362  0.63425
mom_work2    2.99516  2.85462 -0.69479  2.96098  6.66081
mom_work3    5.49666  3.35987  1.33304  5.52975  9.73923
mom_work4    1.42376  2.52727 -1.80580  1.44689  4.62708
mom_age      0.22072  0.32862 -0.19735  0.22120  0.63819
sigma       18.17766  0.63036 17.40984 18.15267 19.00311
```

```
Fit Diagnostics:
           mean     sd       10%      50%      90%   
mean_PPD 86.82546  1.24359 85.22575 86.83978 88.38941

The mean_ppd is the sample average posterior predictive distribution of the outcome variable

MCMC diagnostics
              mcse    Rhat    n_eff
(Intercept)   0.15337 1.00023 3713 
mom_hs1       0.03743 1.00010 3762 
mom_iq        0.00103 0.99989 3602 
mom_work2     0.05694 1.00067 2514 
mom_work3     0.06262 1.00151 2879 
mom_work4     0.05353 1.00093 2229 
mom_age       0.00491 1.00131 4477 
sigma         0.00931 0.99953 4583 
mean_PPD      0.01916 0.99929 4211 
log-posterior 0.04675 1.00031 1891 

For each parameter, mcse is Monte Carlo standard error, n_eff is a crude measure of effective
```

Before we proceed to checking convergence, let us take a look at the prior distributions being used in the above estimation.

```r
prior_summary(model3)
```

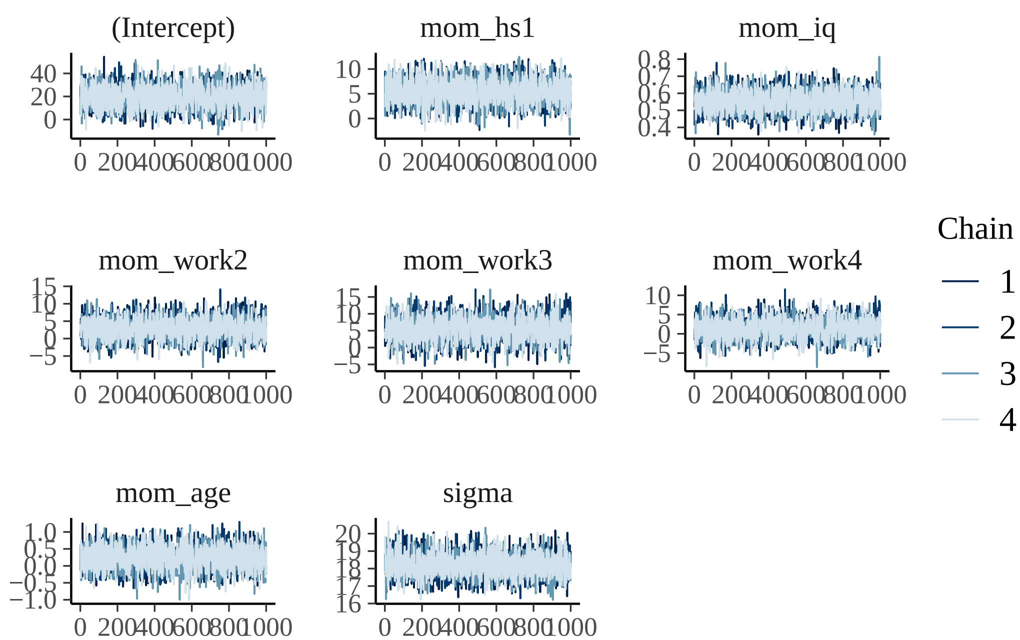
```
Priors for model 'model3' 
------
Intercept (after predictors centered)
  Specified prior:
    ~ normal(location = 87, scale = 2.5)
  Adjusted prior:
    ~ normal(location = 87, scale = 51)

Coefficients
  Specified prior:
    ~ normal(location = [0,0,0,...], scale = [2.5,2.5,2.5,...])
  Adjusted prior:
    ~ normal(location = [0,0,0,...], scale = [124.21,  3.40,122.80,...])

Auxiliary (sigma)
```

```
  Specified prior:
    ~ exponential(rate = 1)
  Adjusted prior:
    ~ exponential(rate = 0.049)
------
See help('prior_summary.stanreg') for more details
```

Checking for sampling quality and convergence

From the output, it can be seen that all $\hat{R}$ values are less than 1.1, the MSCE values are also less than the posterior SD (1.24), and the effective sample sizes are all above 1000. These metrics indicate that the MCMC converge to the same distribution. The trace plot below also shows the same convergence.

```r
plot(model3, "trace")
```

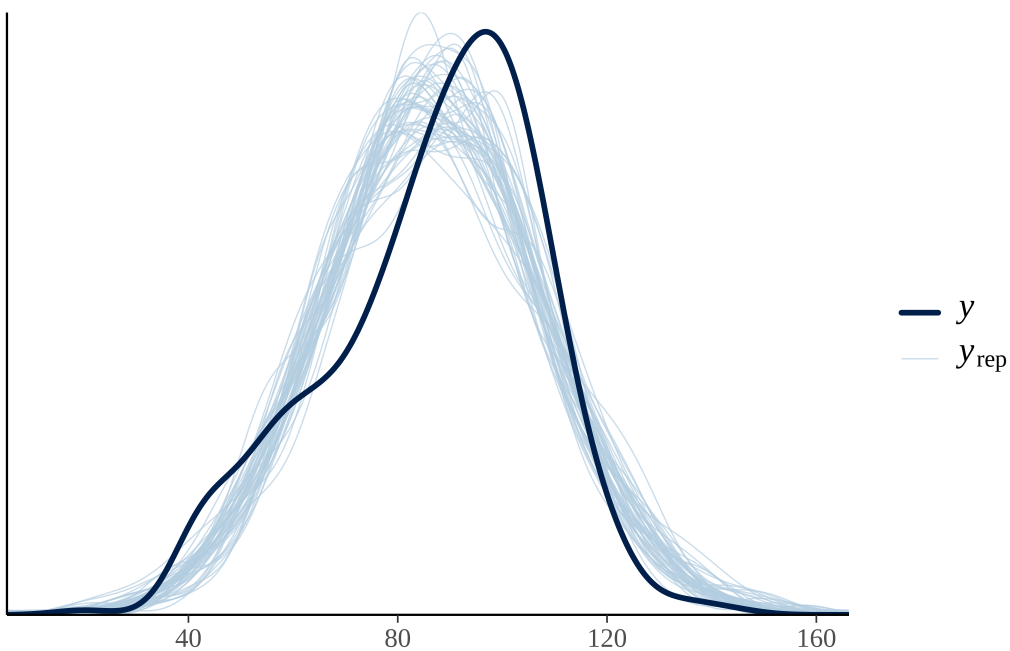


More predictive checks can be requested via the *shinystan()* function.

```r
launch_shinystan(model3)
```

Posterior predictive checking

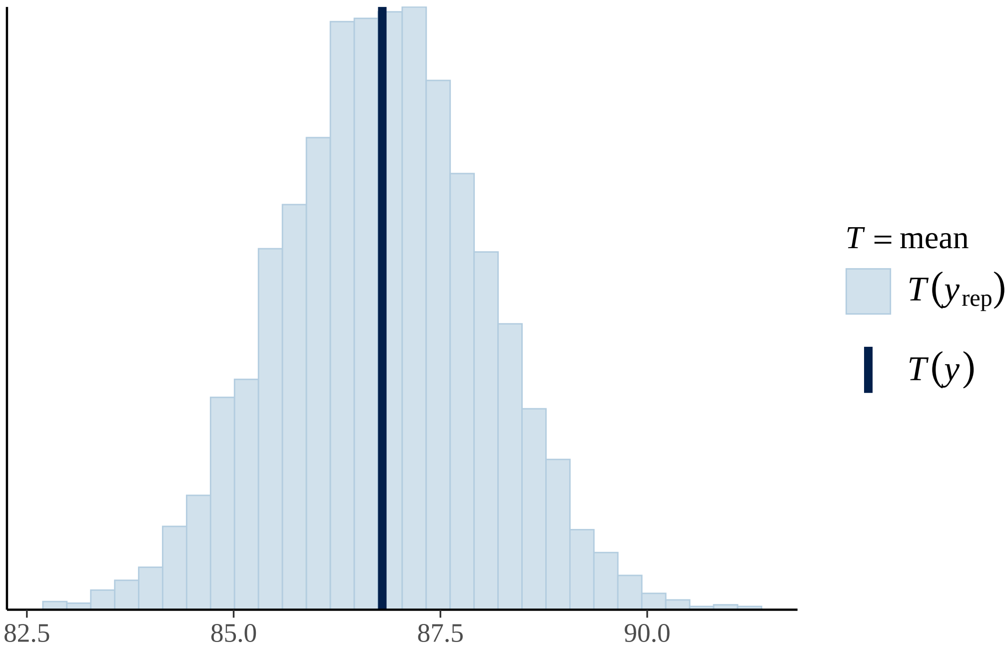
```
pp_check(model3, plotfun = "dens_overlay")
```




```
pp_check(model3,"stat",nreps =100)
```

```
Warning: 'nreps' is ignored for this PPC
```

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



Assuming Model 3 provides a good fit to the data, let us proceed to the interpretation of the estimated coefficients for each predictor. [CLASSROOM EXERCISE!]

Generate predicted outcomes

For example, we might be interested to know the expected cognitive score of a child whose mother is 35 years old, has worked full-time during the first year of the child's life, has no HS education, and has IQ of 95. We can use the posterior distribution to predict this kid's score.

```
newdata <- data.frame(mom_hs = factor(0), mom_iq = 95, mom_work = factor(4), mom_age = 35)
score.pred <- posterior_predict(model3, newdata = newdata)
score.intvl <- predictive_interval(model3, newdata = newdata, prob = 0.95)
print(c(mean(score.pred),score.intvl))
```

```
[1]  81.85102  45.82651 118.72713
```