Lab6

Jun Rao

10/1/2019

Model selection in R and Stan

I have provided you with a data set in which you have measured outcome y and you have measured two input variables $x1$ and $x2$. Your full, most complex linear model that we will consider is:

$$y_i = \alpha + \beta_1 X_{1,i} + \beta_2 X_{2,i}$$

Therefore, we will ignore any multiplicative interactions between the inputs. Importantly, remember from your previous statistics classes that we can condense linear regression into matrix notation: $y = \alpha + \mathbf{BX}$, where $\mathbf{B}$ is a row-vector that holds the slopes and $\mathbf{X}$ is an input matrix of dimensions $(n \times d)$, where n is the number of data points and d is the number of input variables.

Task 1 (15 points)

```
#-------------------------------------
#-------------------------------------
# You may need to install some of these packages
library(tidyverse)
library(rstan)
library(loo)
rstan_options(auto_write = TRUE)
options(mc.cores = parallel::detectCores())
#-------------------------------------
#-------------------------------------
```

1.Import your data (code provided below), and then center and scale your input variables. Center by the mean and scale by 2×standard deviation. This assignment will not work unless you do this step properly. Hint: You must center and scale the two x variable columns separately (i.e., each variable will have its own mean and st. deviation).

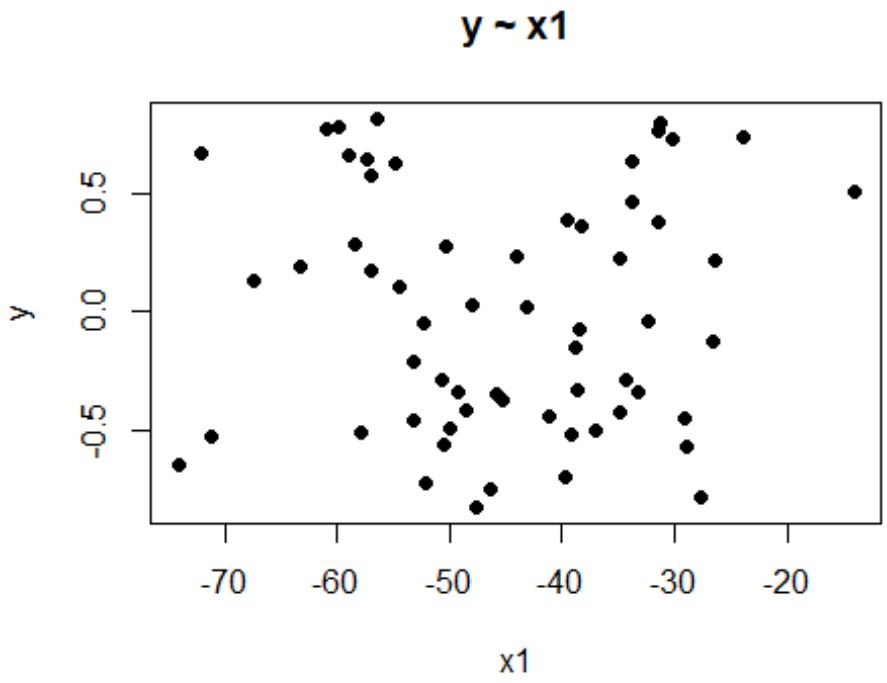
```
# Read in the .CSV file:
data_raw = read_csv("lab6_data.csv")
#observations
n_sample = nrow(data_raw)
n_sample
```

```
## [1] 60
```

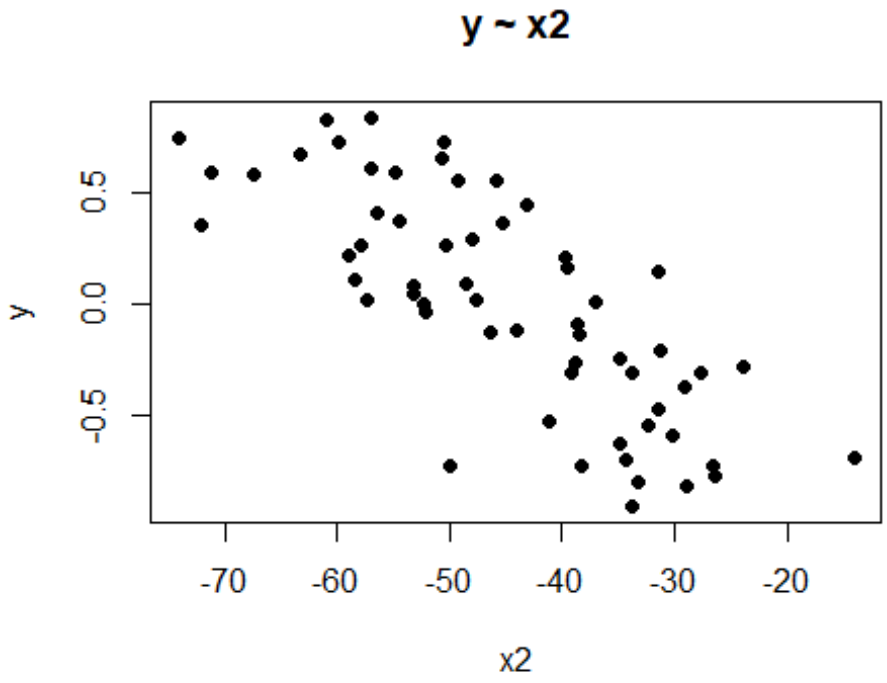
```r
#y
y = data_raw$y

#x1
x1 = data_raw$x1
x1_mean = mean(x1)
x1_sd = sd(x1)
#X2
x2 = data_raw$x2
x2_mean = mean(x2)
x2_sd = sd(x2)

#center and standardize:
x1_scaled = (x1 - x1_mean)/(2*x1_sd)
x2_scaled = (x2 - x2_mean)/(2*x2_sd)
```

2.Create two scatterplots of your outcome versus your two inputs.

```r
plot(y, x1_scaled, main="y ~ x1",xlab="x1", ylab="y", pch=19)
```



```r
plot(y, x2_scaled, main="y ~ x2",xlab="x2", ylab="y", pch=19)
```



3.Look at the provided Stan script, come to an understanding of the data inputs required to run this Stan model. Use the centered and scaled inputs to set up and run the Stan model, by adapting previously provided code. Hint: You will have to monitor the parameter.

```r
stan_data = list(n_input = 2,
                 n_sample = n_sample,
                 x_mat = cbind(x1_scaled,x2_scaled),
                 y_vec = y)

params_monitor = c("beta", "alpha", "sigma", "log_lik")

test_fit = stan(file = "Multi_LinReg.stan",
                data = stan_data,
                pars = params_monitor,
                chains = 1, # How many chains to run
                iter = 10, # How many iterations per chain
                algorithm="NUTS")

##
## SAMPLING FOR MODEL 'Multi_LinReg' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would tak
e 0 seconds.
## Chain 1: Adjust your expectations accordingly!
```

```
## Chain 1:
## Chain 1:
## Chain 1: WARNING: No variance estimation is
## Chain 1:          performed for num_warmup < 20
## Chain 1:
## Chain 1: Iteration: 1 / 10 [ 10%]  (Warmup)
## Chain 1: Iteration: 2 / 10 [ 20%]  (Warmup)
## Chain 1: Iteration: 3 / 10 [ 30%]  (Warmup)
## Chain 1: Iteration: 4 / 10 [ 40%]  (Warmup)
## Chain 1: Iteration: 5 / 10 [ 50%]  (Warmup)
## Chain 1: Iteration: 6 / 10 [ 60%]  (Sampling)
## Chain 1: Iteration: 7 / 10 [ 70%]  (Sampling)
## Chain 1: Iteration: 8 / 10 [ 80%]  (Sampling)
## Chain 1: Iteration: 9 / 10 [ 90%]  (Sampling)
## Chain 1: Iteration: 10 / 10 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0.001 seconds (Warm-up)
## Chain 1:                0 seconds (Sampling)
## Chain 1:                0.001 seconds (Total)
## Chain 1:
```

```
## Now we will run our full model:
# How many samples do we want of each parameter, from each chain?
n_mc_samples = 1000
# How much burn-in?
n_burn = 500
# How much thinning? (take the ith value of the chain)
n_thin = 3
# Total iterations needed:
n_iter_total = (n_mc_samples * n_thin) + n_burn
model_fit =
 stan(fit = test_fit, # So it knows we're already compiled
      file = "Multi_LinReg.stan",
      data = stan_data,
      pars = params_monitor,
      chains = 3,
      warmup = n_burn,
      thin = n_thin,
      iter = n_iter_total,
       algorithm="NUTS")
model_out = rstan::extract(model_fit)
str(model_out)
```

```
## List of 5
##  $ beta   : num [1:3000, 1:2] 7.454 0.995 3.022 3.008 0.847 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ iterations: NULL
##   .. ..$           : NULL
##  $ alpha  : num [1:3000(1d)] -44.3 -44.8 -44.7 -44.4 -45.2 ...
##   ..- attr(*, "dimnames")=List of 1
```

```
##   .. ..$ iterations: NULL
##  $ sigma  : num [1:3000(1d)] 10.26 6.96 8.57 7.74 10.23 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
##  $ log_lik: num [1:3000, 1:60] -3.3 -2.87 -3.13 -3.05 -3.28 ...
##   ..- attr(*, "dimnames")=List of 2
##   .. ..$ iterations: NULL
##   .. ..$           : NULL
##  $ lp__   : num [1:3000(1d)] -163 -161 -158 -159 -161 ...
##   ..- attr(*, "dimnames")=List of 1
##   .. ..$ iterations: NULL
```

4.Show me that your model has converged.

```
# Check Convergence:
tail(summary(model_fit)$summary[, "Rhat"],1)
```

```
##     lp__
## 1.000437
```

Since the value at here less than 1.1, hence we can conclude that the model is converged

5. What are the median and 95% credible intervals for each model parameter?

```
#the median
summary(model_fit)$summary[1:4, "50%"]
```

```
##    beta[1]    beta[2]      alpha      sigma
##   1.364938 -20.242058 -44.960697   8.488155
```

```
#the 95% credible intervals
summary(model_fit)$summary[1:4, "2.5%"]
```

```
##    beta[1]    beta[2]      alpha      sigma
##  -3.079679 -24.527465 -47.117754   7.145730
```

```
summary(model_fit)$summary[1:4, "97.5%"]
```

```
##    beta[1]    beta[2]      alpha      sigma
##   5.805474 -15.815826 -42.934438  10.313266
```

Task 2 (30 points)

1.Re-run your model two more times to store the nested model versions (e.g., with only input 1, and then with only input 2). Hint: You must store these models in objects with different names, however, you will not have to re-write the Stan model at all. Just think about how the inputs might change.

```
#Only input1
stan_data_1 = list(n_input = 1,
                   n_sample = n_sample,
                   x_mat = cbind(x1_scaled),
                   y_vec = y)
#Only input2
stan_data_2 = list(n_input = 1,
                   n_sample = n_sample,
                   x_mat = cbind(x2_scaled),
                   y_vec = y)

params_monitor = c("beta", "alpha", "sigma", "log_lik")

test_fit_1 = stan(file = "Multi_LinReg.stan",
                  data = stan_data_1,
                  pars = params_monitor,
                  chains = 1, # How many chains to run
                  iter = 10, # How many iterations per chain
                  algorithm="NUTS")

##
## SAMPLING FOR MODEL 'Multi_LinReg' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would tak
e 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: No variance estimation is
## Chain 1:          performed for num_warmup < 20
## Chain 1:
## Chain 1: Iteration: 1 / 10 [ 10%]  (Warmup)
## Chain 1: Iteration: 2 / 10 [ 20%]  (Warmup)
## Chain 1: Iteration: 3 / 10 [ 30%]  (Warmup)
## Chain 1: Iteration: 4 / 10 [ 40%]  (Warmup)
## Chain 1: Iteration: 5 / 10 [ 50%]  (Warmup)
## Chain 1: Iteration: 6 / 10 [ 60%]  (Sampling)
## Chain 1: Iteration: 7 / 10 [ 70%]  (Sampling)
## Chain 1: Iteration: 8 / 10 [ 80%]  (Sampling)
## Chain 1: Iteration: 9 / 10 [ 90%]  (Sampling)
## Chain 1: Iteration: 10 / 10 [100%]  (Sampling)
## Chain 1:
```

```
## Chain 1:  Elapsed Time: 0.001 seconds (Warm-up)
## Chain 1:                0.003 seconds (Sampling)
## Chain 1:                0.004 seconds (Total)
## Chain 1:
```

```
test_fit_2 = stan(file = "Multi_LinReg.stan",
                  data = stan_data_2,
                  pars = params_monitor,
                  chains = 1, # How many chains to run
                  iter = 10, # How many iterations per chain
                  algorithm="NUTS")
```

```
##
## SAMPLING FOR MODEL 'Multi_LinReg' NOW (CHAIN 1).
## Chain 1:
## Chain 1: Gradient evaluation took 0 seconds
## Chain 1: 1000 transitions using 10 leapfrog steps per transition would tak
e 0 seconds.
## Chain 1: Adjust your expectations accordingly!
## Chain 1:
## Chain 1:
## Chain 1: WARNING: No variance estimation is
## Chain 1:          performed for num_warmup < 20
## Chain 1:
## Chain 1: Iteration: 1 / 10 [ 10%]  (Warmup)
## Chain 1: Iteration: 2 / 10 [ 20%]  (Warmup)
## Chain 1: Iteration: 3 / 10 [ 30%]  (Warmup)
## Chain 1: Iteration: 4 / 10 [ 40%]  (Warmup)
## Chain 1: Iteration: 5 / 10 [ 50%]  (Warmup)
## Chain 1: Iteration: 6 / 10 [ 60%]  (Sampling)
## Chain 1: Iteration: 7 / 10 [ 70%]  (Sampling)
## Chain 1: Iteration: 8 / 10 [ 80%]  (Sampling)
## Chain 1: Iteration: 9 / 10 [ 90%]  (Sampling)
## Chain 1: Iteration: 10 / 10 [100%]  (Sampling)
## Chain 1:
## Chain 1:  Elapsed Time: 0 seconds (Warm-up)
## Chain 1:                0 seconds (Sampling)
## Chain 1:                0 seconds (Total)
## Chain 1:
```

```
## Now we will run our full model:
# How many samples do we want of each parameter, from each chain?
n_mc_samples = 1000
# How much burn-in?
n_burn = 500
# How much thinning? (take the ith value of the chain)
n_thin = 3
# Total iterations needed:
n_iter_total = (n_mc_samples * n_thin) + n_burn
```

```
model_fit_1 =
 stan(fit = test_fit_1, # So it knows we're already compiled
      file = "Multi_LinReg.stan",
      data = stan_data_1,
      pars = params_monitor,
      chains = 3,
      warmup = n_burn,
      thin = n_thin,
      iter = n_iter_total,
      algorithm="NUTS")

model_fit_2 =
 stan(fit = test_fit_2, # So it knows we're already compiled
      file = "Multi_LinReg.stan",
      data = stan_data_2,
      pars = params_monitor,
      chains = 3,
      warmup = n_burn,
      thin = n_thin,
      iter = n_iter_total,
      algorithm="NUTS")

tail(summary(model_fit_1)$summary[, "Rhat"],1)

##     lp__
## 0.999492

tail(summary(model_fit_2)$summary[, "Rhat"],1)

##     lp__
## 1.001422

#the median
summary(model_fit_1)$summary[1:3, "50%"]

##      beta[1]        alpha        sigma
##  -0.01201261 -44.95451231  13.18712198

summary(model_fit_2)$summary[1:3, "50%"]

##   beta[1]     alpha     sigma
## -20.23118 -45.01087   8.40686

#the 95% credible intervals
summary(model_fit_1)$summary[1:3, "2.5%"]

##    beta[1]      alpha      sigma
##  -6.890235 -48.378096  11.128468

summary(model_fit_1)$summary[1:3, "97.5%"]
```

```
##    beta[1]      alpha      sigma
##   6.917912 -41.619817  15.897878
```

```
#the 95% credible intervals
summary(model_fit_2)$summary[1:3, "2.5%"]
```

```
##    beta[1]      alpha      sigma
## -24.407160 -47.182270   7.070992
```

```
summary(model_fit_2)$summary[1:3, "97.5%"]
```

```
##   beta[1]     alpha     sigma
## -15.88004 -42.86738  10.19127
```

2.Use the () function to calculate the LOO-IC for each model. Follow my example code below, and make sure to store these into separate objects for each model run.

```
# Because eval = FALSE this code chunk will not run.
# You will have to input your correct objects into this function.

# Calculate the LOO and WAIC for a single model:
# I'm doing this for the 'full' or most complex model
log_lik_full = extract_log_lik(model_fit)
log_lik_1 = extract_log_lik(model_fit_1)
log_lik_2 = extract_log_lik(model_fit_2)

loo_full = loo(log_lik_full)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
loo_sub1 = loo(log_lik_1)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
loo_sub2 = loo(log_lik_2)
```

```
## Warning: Relative effective sample sizes ('r_eff' argument) not specified.
## For models fit with MCMC, the reported PSIS effective sample sizes and
## MCSE estimates will be over-optimistic.
```

```
waic_full = waic(log_lik_full)
```

```
## Warning: 2 (3.3%) p_waic estimates greater than 0.4. We recommend trying
## loo instead.
```

```
waic_sub1 = waic(log_lik_1)
waic_sub2 = waic(log_lik_2)
```

```
## Warning: 1 (1.7%) p_waic estimates greater than 0.4. We recommend trying
## loo instead.
```

```
loo_full
```

```
##
## Computed from 3000 by 60 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -216.1  5.3
## p_loo         4.0  0.8
## looic       432.3 10.7
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
loo_sub1
```

```
##
## Computed from 3000 by 60 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -242.2  5.0
## p_loo         3.0  0.6
## looic       484.5 10.0
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
loo_sub2
```

```
##
## Computed from 3000 by 60 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -215.2  5.4
## p_loo         2.9  0.7
## looic       430.3 10.7
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```
waic_full
```

```
##
## Computed from 3000 by 60 log-likelihood matrix
```

```
##
##           Estimate   SE
## elpd_waic   -216.1  5.3
## p_waic         3.9  0.8
## waic         432.2 10.6

## Warning: 2 (3.3%) p_waic estimates greater than 0.4. We recommend trying
## loo instead.

waic_sub1

##
## Computed from 3000 by 60 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic   -242.2  5.0
## p_waic         3.0  0.6
## waic         484.4 10.0

waic_sub2

##
## Computed from 3000 by 60 log-likelihood matrix
##
##           Estimate   SE
## elpd_waic   -215.2  5.4
## p_waic         2.9  0.7
## waic         430.3 10.7

## Warning: 1 (1.7%) p_waic estimates greater than 0.4. We recommend trying
## loo instead.
```

3.Use the () function to compare all three of your nested models using the LOO-CV. Your code will look something like:

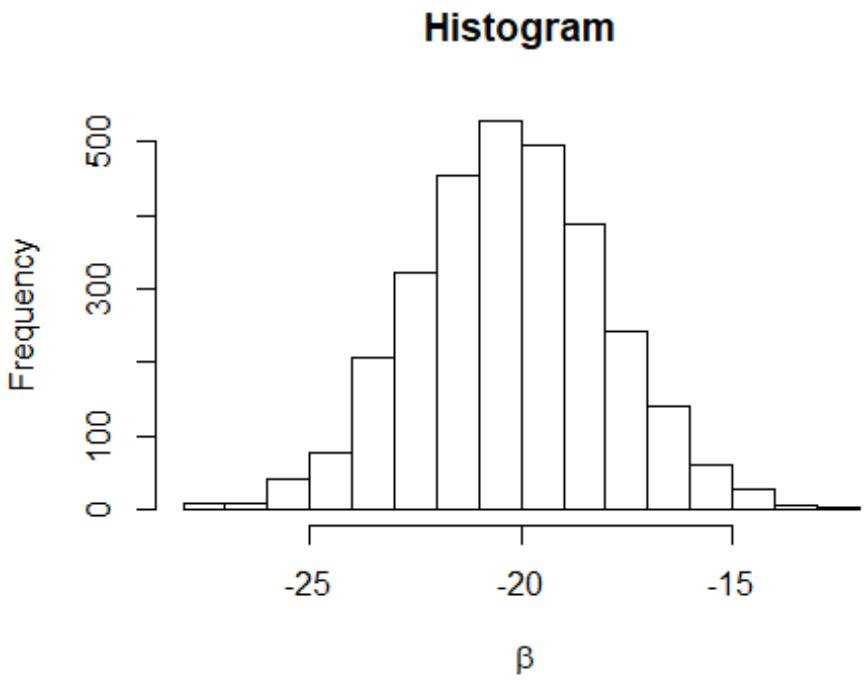
```
# Because eval = FALSE this code chunk will not run.
# You will have to input your correct objects into this function.

loo::compare(loo_full, loo_sub1, loo_sub2)

##          elpd_diff se_diff elpd_loo p_loo  looic
## loo_sub2    0.0       0.0  -215.2      2.9  430.3
## loo_full   -1.0       0.7  -216.1      4.0  432.3
## loo_sub1  -27.1       4.9  -242.2      3.0  484.5
```

4. Give a brief interpretation of your results of this statistical analysis and model comparison, citing the quantitative metrics as support for your conclusions. Remember your goal is to choose the most parsimonious model.

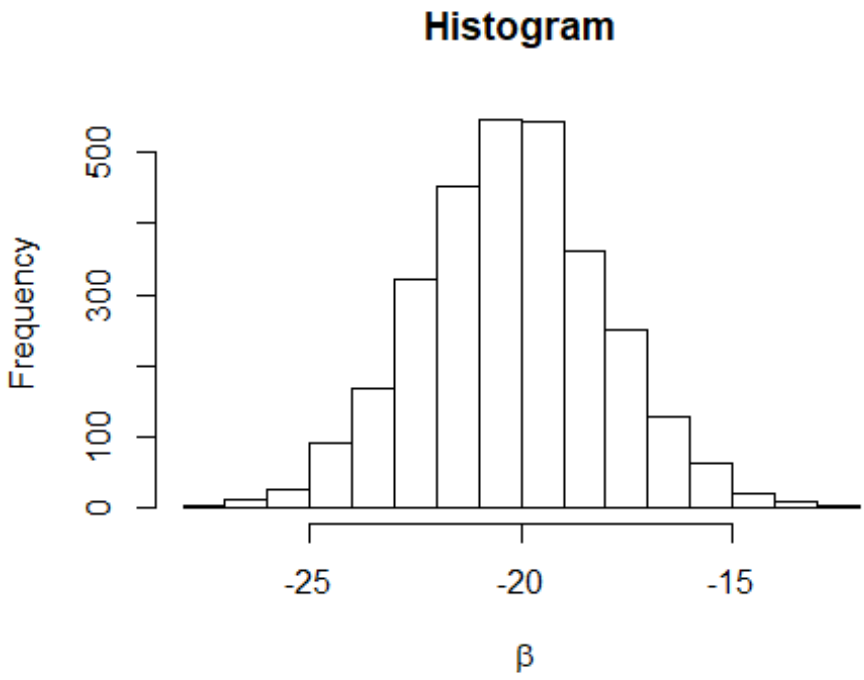
According to the result shown in the question 3, we may could conclude that the best model is model_fit_2, which only have the 2nd input. The value of elpd_loo is -215.1. Compare to the full model, it is also simpler (because we only have one input value) even the has the close elpd_loo value.

5.Show me visually and quantitatively that the variance of the posterior samples of β_2 from the most complex model is than the variance of the same estimate from the simpler model. The difference will be subtle, but you can imagine that if you had many input variables the effect would become compounded.

```
hist(extract(model_fit)$beta[,2], main = paste("Histogram"),xlab=expression(~
beta))
```




```
hist(extract(model_fit_2)$beta, main = paste("Histogram"),xlab=expression(~be
ta))
```




```
var(extract(model_fit)$beta[,2])
```

```
## [1] 5.113633
```

```
var(extract(model_fit_2)$beta)
```

```
##          [,1]
## [1,] 4.843807
```