Data 582 - Bayesian Inference

Lab 5: Stan

Contents

1 Introduction **1**

2 Examples **2**

2.1 Toy Model 2

2.2 Beta-Binomial 6

2.3 Linear Regression 9

3 References **14**

1 Introduction

As a final consideration in our Bayesian module let's conduct MCMC simulation using the rstan package; R's interface to Stan. There are two essential steps to all rstan analyses:

1. define the Bayesian model structure in rstan notation
2. simulate the posterior.

Stan models are written in their own syntax. The flexible model has the form as given below which can store to a text file, ex *mymodel.stan*.

```
data{
  #... a data block
}

parameters{
  # ... a parameter block
}

model {
  # ... a model block
}
```

Before we begin, make sure you load the appropriate libraries:

```r
# Load packages
library(bayesplot)
library(tidyverse)
library(rstan)
library(janitor)
library(bayesrules)
```

If you have enough RAM, you can go ahead an use the recommendations provided by rstan when you load the pacakge. For example, you can call to all available cores and run multiple chains in parallel by using:

```r
options(mc.cores = parallel::detectCores())
```

We will look at a few familiar models to gain some familiarity with the key elements of an rstan simulation.

2 Examples

2.1 Toy Model

Let's go through the example on the Rstan YouTube channel (found here). We begin by generating a thousand data points from a Normal(5,1).

```r
set.seed(3249) # for reproducibility
X = rnorm(1000, 5,1)
```

Suppose we want to build a Bayesian model in Stan for recovering the parameters μ and σ from only the X data values. To do this, store the following text to a file called "mymodel.stan". Recall that you can create Stan files directly in RStudio by navigating to **New File** $\rightarrow$ **Stan File**

```
// saved as mymodel.stan
data {
  int N;        // number of data points
  real X[N];    // data values
}

parameters {
  real mu;       // mean
  real sigma;    // standard deviation
}
```

```
model {
  X ~ normal(mu, sigma);
}
```

ALERT: Be sure that your Stan programs ends in a blank line without any characters including spaces and comments. Make sure this file is saved in your working directory and then run the following.

Data will be needed to be stored in a named list (with names corresponding to the variable names you specified in your data block in your stan model). So let's prepare the data in list format:

```
mydata = list(N= length(X), X=X)
```

Now to run the model it's as simple as:

```
fit = stan(file="mymodel.stan", data = mydata)
```

To see how well our model performed we can type:

```
print(fit) # mean, sd, of mu and sigma

## Inference for Stan model: mymodel.
## 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 Rhat
## mu       5.10    0.00 0.03    5.04    5.08    5.10    5.12    5.16  3574    1
## sigma    0.98    0.00 0.02    0.94    0.97    0.98    1.00    1.03  3419    1
## lp__  -483.65    0.02 0.98 -486.31 -484.00 -483.34 -482.97 -482.69  1845    1
##
## Samples were drawn using NUTS(diag_e) at Sun Apr 24 13:02:56 2022.
## 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).
```

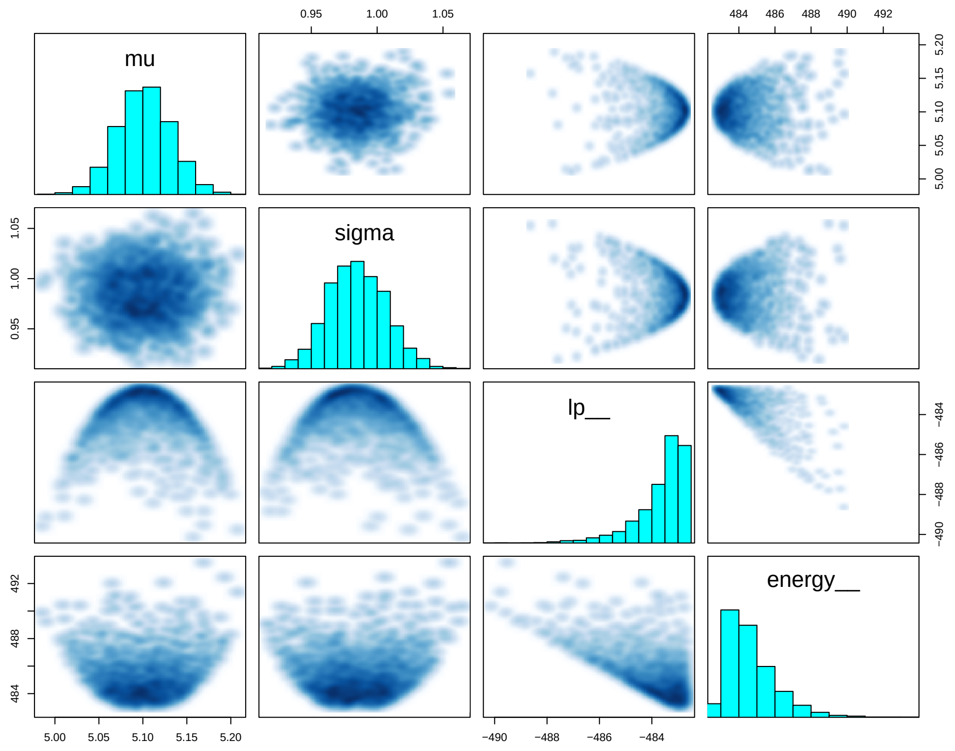
Alternatively we can use the following for more detailed statistics:

```
summary(fit) # for more detailed statistics

## $summary
##                mean      se_mean         sd         2.5%         25%         50%
## mu        5.1013670 0.0005128070 0.03065815    5.0417295    5.080776    5.1014557
## sigma     0.9847055 0.0003742886 0.02188506    0.9429688    0.969242    0.9842833
## lp__   -483.6468653 0.0227650270 0.97795088 -486.3149259 -483.997098 -483.3386071
##                75%       97.5%    n_eff      Rhat
## mu       5.1223339    5.162365 3574.243 0.9998848
## sigma    0.9999984    1.028704 3418.869 1.0005833
## lp__  -482.9703468 -482.694106 1845.431 0.9996914
##
## $c_summary
## , , chains = chain:1
##
##          stats
## parameter         mean         sd         2.5%         25%          50%          75%
##     mu       5.1006113 0.03066521    5.0436322    5.078948    5.1009061    5.1217554
##     sigma    0.9846388 0.02106767    0.9444487    0.969999    0.9838435    0.9988903
##     lp__  -483.6099956 0.91644530 -486.2207142 -483.975314 -483.3386071 -482.9549861
##          stats
## parameter       97.5%
##     mu       5.159847
##     sigma    1.027254
##     lp__  -482.691111
##
## , , chains = chain:2
##
##          stats
## parameter         mean         sd         2.5%          25%          50%          75%
##     mu       5.1019390 0.02983002    5.0437689    5.0828585    5.1019493    5.1211776
##     sigma    0.9841204 0.02228696    0.9425717    0.9686214    0.9836843    0.9999975
##     lp__  -483.6392559 0.99898506 -486.3953017 -483.9408512 -483.3217969 -482.9666844
##          stats
## parameter       97.5%
##     mu       5.160218
##     sigma    1.028404
##     lp__  -482.692659
##
## , , chains = chain:3
##
##          stats
## parameter         mean         sd        2.5%          25%          50%         75%
##     mu       5.1007694 0.03028056    5.041290    5.0808273    5.1006535    5.120803
##     sigma    0.9856104 0.02252161    0.943165    0.9701831    0.9853211    1.001789
##     lp__  -483.6640772 1.01246262 -486.347808 -484.0516920 -483.3407087 -482.982068
##          stats
## parameter       97.5%
##     mu       5.161980
##     sigma    1.028706
##     lp__  -482.696465
```

```
##
## , , chains = chain:4
##
##          stats
## parameter         mean          sd         2.5%          25%         50%         75%
##     mu        5.1021484 0.03183625    5.0392817    5.0794910    5.102406    5.124741
##     sigma     0.9844526 0.02163917    0.9421689    0.9687273    0.984484    0.999236
##     lp__   -483.6741325 0.98134631 -486.2958329 -484.0111565 -483.358678 -482.975042
##          stats
## parameter       97.5%
##     mu        5.162933
##     sigma     1.029551
##     lp__   -482.697515
```

To visualize the posterior draws of the parameters and the marginalized posteriors we could use:

```
pairs(fit)
```



To extract the values of the chain, i.e. the posterior draws of the parameters, we use, well `extract`...

```
params = extract(fit)
params$mu[1:10]

##  [1] 5.090381 5.109742 5.112172 5.091521 5.083026 5.150670 5.159669 5.074334 5.081887
## [10] 5.118243

params$mu[1:10]

##  [1] 5.090381 5.109742 5.112172 5.091521 5.083026 5.150670 5.159669 5.074334 5.081887
## [10] 5.118243
```

You can save the fitted model using

```
save(fit, file = "myfit")
# load(fit) # (to load the fit back into a new R sesssion)
```

2.2 Beta-Binomial

As discussed in lecture, we can define a Beta-Binomial with the following specs:

$$\begin{aligned} Y|\theta &\sim \text{Bin}(10, \pi) \\ \theta &\sim \text{Beta}(2, 2). \end{aligned}$$

through the following two-step process:

Step 1: Define the stan model. Recall this must be a *character string* stored to anR object or saved to a .stan file (you can create this in R by navigating to **New File** $\rightarrow$ **Stan File**)

```
# STEP 1: DEFINE the model
bb_model <- "
  data {
    int<lower = 0, upper = 10> y;
  }
  parameters {
    real<lower = 0, upper = 1> theta;
  }
  model {
    y ~ binomial(10, theta);
    theta ~ beta(2, 2);
  }
"
```

Step 2: Simulate the posterior

```
# STEP 2: SIMULATE the posterior
bb_sim <- stan(model_code = bb_model, data = list(y = 9),
               chains = 4, iter = 5000*2, seed = 84735)
```

The result, stored in `bb_sim`, is a stanfit object (see `?stanfit`). This object includes four parallel Markov chains run for 10,000 iterations each. After tossing out the first 5,000 iterations of all four chains, we end up with four separate Markov chain samples of size 5,000, or a combined Markov chain sample size of 20,000.

We can extract the chains at least two different ways. The following stores the long chain comprised the four chains appended together:

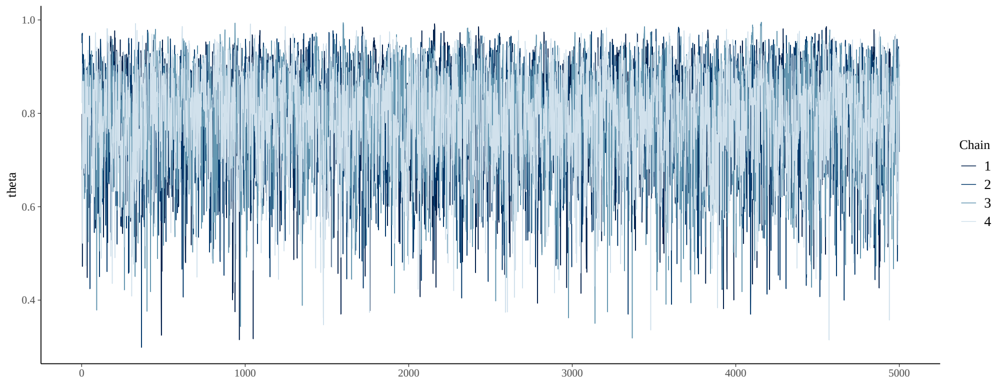
```
samp <- extract(bb_sim)
theta.chain = samp$theta
length(theta.chain)

## [1] 20000

head(theta.chain)

## [1] 0.7787525 0.7729890 0.8971423 0.9233372 0.8273319 0.9054819
```

If you want to look at the four chains separately you could save them using:

```
theta.chains = as.array(bb_sim, pars = "theta")
dim(theta.chains)

## [1] 5000    4    1

head(theta.chains)

## , , parameters = theta
##
##             chains
## iterations   chain:1   chain:2   chain:3   chain:4
##       [1,] 0.6830525 0.9505094 0.6788978 0.8220786
##       [2,] 0.7105128 0.9700015 0.7318025 0.9061504
##       [3,] 0.7984070 0.9663070 0.7518239 0.8572139
##       [4,] 0.4720030 0.9717269 0.7975232 0.8683416
##       [5,] 0.7221846 0.7613560 0.7801351 0.5329752
##       [6,] 0.8308741 0.7597380 0.8672630 0.5214215
```

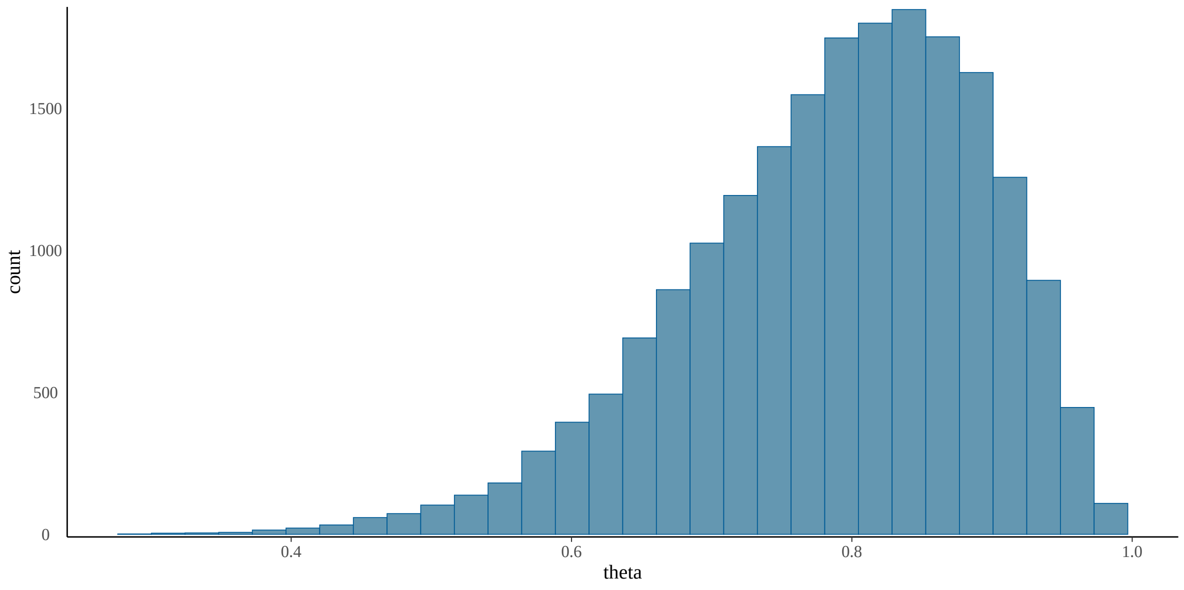
To see the traceplots we could use:

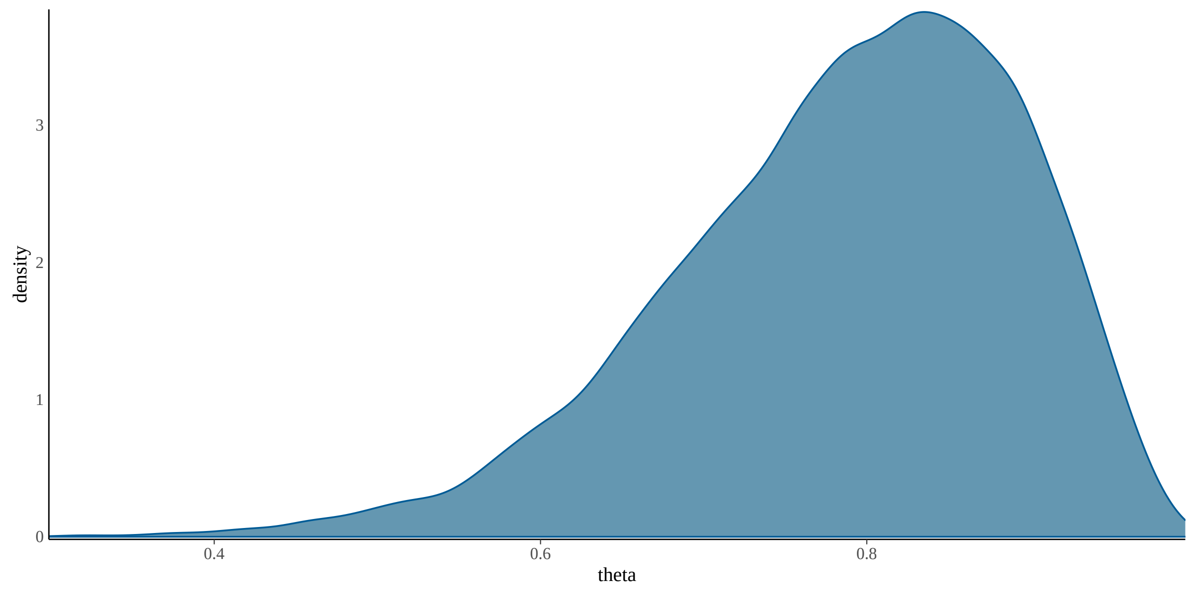
```
mcmc_trace(bb_sim, pars = "theta")
```



The histogram and density plot in below provide a snapshot of this distribution for the combined 20,000 chain values (5,000 from each of the four separate chains).

```
# Histogram of the Markov chain values
mcmc_hist(bb_sim, pars = "theta") +
  yaxis_text(TRUE) +
  ylab("count")

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

# Density plot of the Markov chain values
mcmc_dens(bb_sim, pars = "theta") +
  yaxis_text(TRUE) +
  ylab("density")
```





Although we haven't talked about much of the outputs that you can obtain using the following function, I encourage you to play around with the outputs of the following (it will open in your default browswer)

```r
library(shinystan)
launch_shinystan(bb_sim)
```

2.3 Linear Regression

Bike Data

Below is the stan model for performing the Normal regression model we discussed in lecture regarding the rideshare in Washington D.C.

```r
# STEP 1: DEFINE the model
stan_bike_model <- "
  data {
    int<lower = 0> n;
    vector[n] Y;
    vector[n] X;
  }
  parameters {
    real beta0;
    real beta1;
    real<lower = 0> sigma;
  }
  model {
    Y ~ normal(beta0 + beta1 * X, sigma);
    beta0 ~ normal(-2000, 1000);
    beta1 ~ normal(100, 40);
    sigma ~ exponential(0.0008);
  }
"
```

Note that the sampling statement in this model is vectorized meaning

```
Y ~ normal(beta0 + beta1 * X, sigma);
```

applies to all *y*s in the `Y` vector. Alternatively, we could have specified it using:

```
for (n in 1:N)
  y[n] ~ normal(alpha + beta * x[n], sigma);
```

For **step 2** we can prepare our data by putting them in a list with the same names as the variables declare in the stan model:

```
data(bikes)
bikedata = list(n = nrow(bikes), Y = bikes$rides, X = bikes$temp_feel)
```

which will be fed into the stan() function:

```
stan_bike_sim <-
  stan(model_code = stan_bike_model,
       data = bikedata,
       chains = 4, iter = 5000*2, seed = 84735)
```

If you prefer, you can simply create this list when you execute the function.

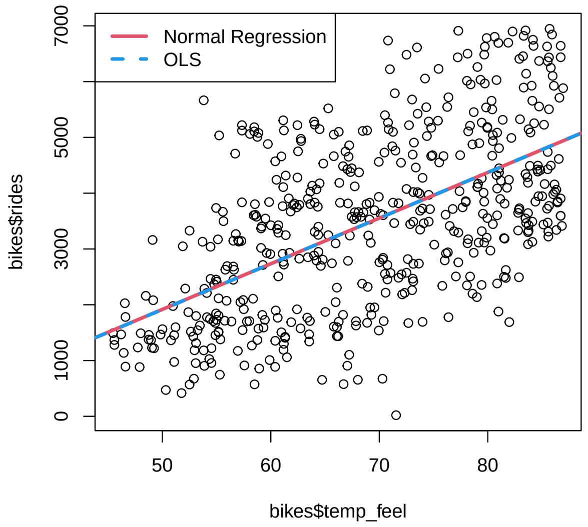
```
# STEP 2: SIMULATE the posterior
stan_bike_sim <-
  stan(model_code = stan_bike_model,
       data = list(n = nrow(bikes), Y = bikes$rides, X = bikes$temp_feel),
       chains = 4, iter = 5000*2, seed = 84735)
```

As before, we can see how stan did by printing the fit:

```
print(stan_bike_sim)

## Inference for Stan model: 45eca288556576b95d342a6c16ab73b2.
## 4 chains, each with iter=10000; warmup=5000; thin=1;
## post-warmup draws per chain=5000, total post-warmup draws=20000.
##
##             mean se_mean     sd     2.5%      25%      50%      75%    97.5% n_eff Rhat
## beta0 -2174.63    4.35 334.83 -2831.78 -2400.82 -2180.68 -1951.81 -1509.02  5931    1
## beta1    81.82    0.06   4.79    72.32    78.63    81.89    85.05    91.26  5989    1
## sigma  1282.51    0.43  40.42  1205.98  1255.08  1281.37  1309.00  1365.74  9035    1
## lp__   -3822.29    0.02   1.22 -3825.46 -3822.85 -3821.97 -3821.39 -3820.89  6348    1
##
## Samples were drawn using NUTS(diag_e) at Tue Apr 26 13:10:08 2022.
## 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).
```

If we wanted to we could plot a single fitted line based on the mean of the best fit line parameter. For fun, let's compare that to the Frequentist OLS line

```
plot(bikes$temp_feel, bikes$rides)
params = extract(stan_bike_sim)
bhat0 = mean(params$beta0)
bhat1 = mean(params$beta1)
abline(a = bhat0, b = bhat1, col = 2, lwd = 3)
olsfit <- lm(rides~temp_feel, data = bikes)
abline(olsfit, col = 4, lty=2, lwd = 3)
legend("topleft", legend = c("Normal Regression", "OLS"), col =c(2,4), lwd = 3, lty = c(
```



Comparing the two methods you can see that the lines are identical. Here are the parameter values:

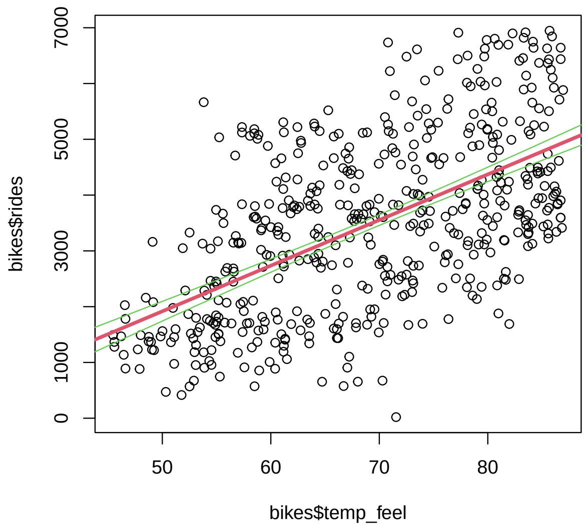
```
olsfit

##
```

```
## Call:
## lm(formula = rides ~ temp_feel, data = bikes)
##
## Coefficients:
## (Intercept)    temp_feel
##    -2179.27        81.88
```

```
c(bhat0, bhat1)

## [1] -2174.63217    81.82381
```

you can see that the lines are very similar. We can also obtain a posterior interval using all of the posterior draws of the parameters and computing the 95 percentile across the range of X values.

```
plot(bikes$temp_feel, bikes$rides)
abline(a = bhat0, b = bhat1, col = 2, lwd = 3)

# trick to getting the plotting range for x
mylims <- par("usr")
xr = seq(mylims[1], mylims[2], length=100)
yCI = sapply(xr, function(k) quantile(params$beta1*k + params$beta0, probs = c(0.05, 0.9
lines(xr, yCI[1,], col = 3)
lines(xr, yCI[2,], col = 3)
```



Iris data

We can really easily use this same model with different data. For instance let's run it on the iris data set with Sepal.Length of the versicolor species as our predictor and Petal.Length of the versicolor species as our response. We do need to adjust our priors however:

```
# STEP 1: DEFINE the model
stan_iris_model <- "
  data {
    int<lower = 0> n;
    vector[n] Y;
    vector[n] X;
  }
  parameters {
    real beta0;
    real beta1;
    real<lower = 0> sigma;
  }
  model {
    Y ~ normal(beta0 + beta1 * X, sigma);
    beta0 ~ normal(0, 10);
    beta1 ~ normal(0, 10);
    sigma ~ normal(0, 1);
  }
"
```

Following the example from the Rstan youTube channel (here) notice that they specify a normal prior for σ. Since they declared their lower bound for sigma in the parameters block this essentially just truncates the normal distribution to only consider positive values.

```
data(iris)
# create a subset of the data just consisting of the veriscolor species:
versicolor = subset(iris, Species=="versicolor")
irisdata = list(n = nrow(versicolor), Y = versicolor$Petal.Length,
                X = versicolor$Sepal.Length)
```

and plugging into stan

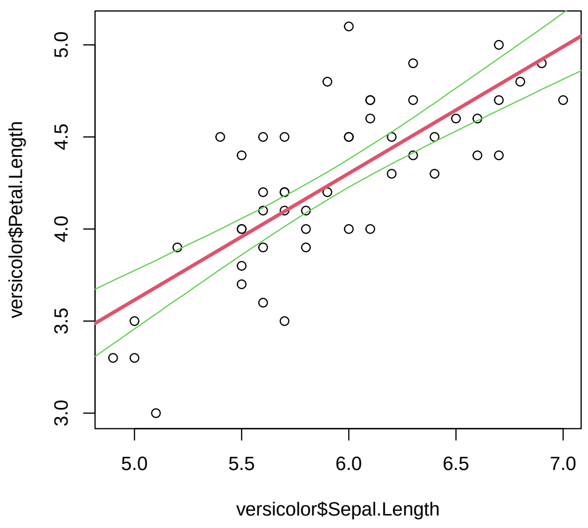
```
stan_iris_sim <-
  stan(model_code = stan_iris_model,
       data = irisdata, seed = 84735)
```

As before, we can see how stan did by printing the fit:

```r
print(stan_iris_sim)

## Inference for Stan model: c2784ab3e61bea9abff5adf0a372158c.
## 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 Rhat
## beta0  0.18    0.02 0.54 -0.95 -0.18  0.18  0.53  1.26  1128    1
## beta1  0.69    0.00 0.09  0.51  0.63  0.69  0.75  0.88  1127    1
## sigma  0.32    0.00 0.03  0.26  0.30  0.32  0.34  0.39  1459    1
## lp__  31.47    0.04 1.32 27.96 30.87 31.82 32.43 32.94   979    1
##
## Samples were drawn using NUTS(diag_e) at Sun Apr 24 15:21:26 2022.
## 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).
```

Plotting the fit with the 95 percentile across the range of X values:

```r
plot(versicolor$Sepal.Length, versicolor$Petal.Length)
params = extract(stan_iris_sim)
bhat0 = mean(params$beta0)
bhat1 = mean(params$beta1)
abline(a = bhat0, b = bhat1, col = 2, lwd = 3)
mylims <- par("usr")
xr = seq(mylims[1], mylims[2], length=100)
yCI = sapply(xr, function(k) quantile(params$beta1*k + params$beta0, probs = c(0.05, 0.9
lines(xr, yCI[1,], col = 3)
lines(xr, yCI[2,], col = 3)
```



3 References

`https://mc-stan.org/users/documentation/` Stan's User Guide Stan's Reference Manual