STATISTICAL METHODS FOR DATA SCIENCE PROJECT REPORT
FINAL PROJECT (2016/2017)

Nthimbo Gift Tembo

July 2018

BAYESIAN INFERENCE OF SOFT DRINK DELIVERY TIMES VIA NORMAL REGRESSION MODEL

Table of Contents

1.Data and Model Description

In this project I have considered Normal based bayesian linear regression. The data deals with the quality of delivery times of soft drink company. My interest is in estimation of the required time needed by each employee to refill an automatic vending machine. Data was collected as a result of small quality assurance study suggesting that delivery times are affected by two independent variables. These variables are the number of cases of stocked products that the employee needs to deliver and the distance the employee has to travel to reach the vending machine. Therefore, I have considered four models as follows:

model1: does not support notion of independent variables

model2: suggests that only the number of cases influences the delivery times

model3: suggests that only distance travelled affects delivery times

model4: suggesting that delivery times are influenced by both variables, distance and the number of cases.

Model formulation:

$$Y|x_1,\ldots,x_p \sim N(\mu(\boldsymbol{\beta},x_1,\ldots,x_p),\sigma^2)$$

where

$$\mu(\boldsymbol{\beta},x_1,\ldots,x_p) = \beta_0+\beta_1x_1+,\ldots,\beta_px_p = \beta_0+\sum_{j=1}^{p}\beta_jx_j$$

where σ^2 and $\boldsymbol{\beta} = (\beta_0,\beta_1,..,\beta_p)^T$ are a set of regression parameters of interest.

Likelihood is given by;

$Y = (y_1,\ldots,y_n)$ and $x_{i1},\ldots,x_{ip}$, for values of explanatory variables $X_1,\ldots,X_p$ for $i = 1,\ldots,n$

Prior distribution of parameters:

$$f(\boldsymbol{\beta},\tau) = \prod_{j=0}^{p} f(\beta_j)f(\tau)$$

therefore,

$\beta_j \sim N(\beta_j\mu, c_j^2)\ j = 0,\ldots,p$ and $\tau \sim gamma(a,b)$

c_j^2 is the prior variance and is set to a high number since I have low prior belief about the model.

Here I load data which consists of 25 observations and make simple plots to see the data

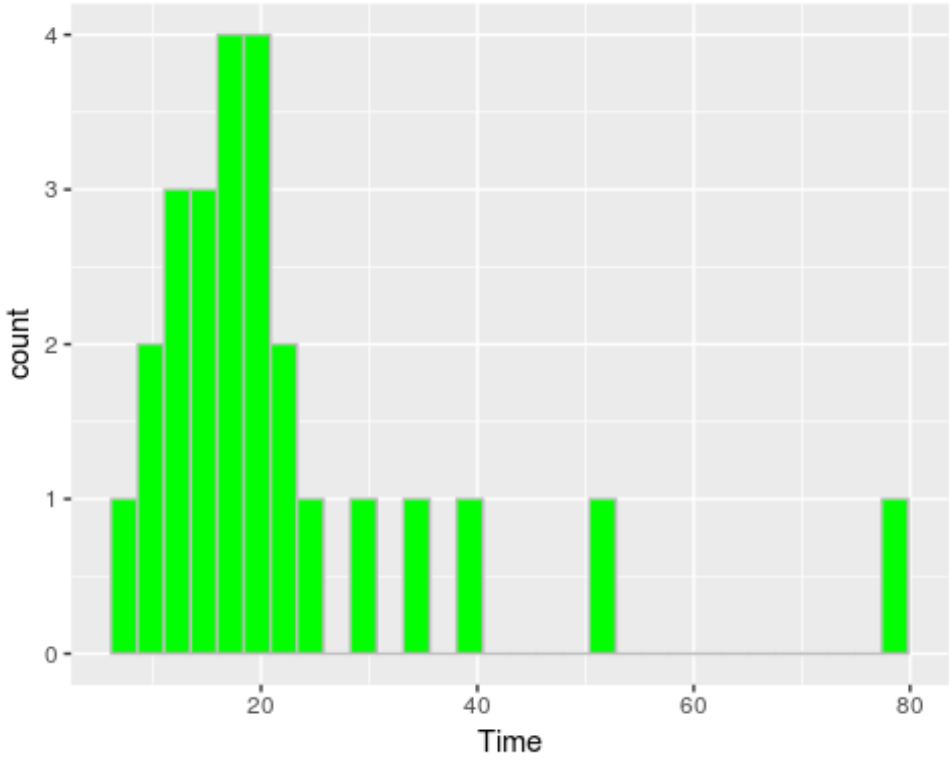
```
#load the softdrinks data
my.data <- read.csv("softdrinks.txt", sep = "")
```

```r
#get sample of our data
head(my.data)
```

```
##    Time Cases Distance
## 1 16.68     7      560
## 2 11.50     3      220
## 3 12.03     3      340
## 4 14.88     4       80
## 5 13.75     6      150
## 6 18.11     7      330
```

```r
#plot of the response(y) of the data
ggplot(my.data,aes(Time)) + geom_histogram(fill = "green", color = "gray")
```

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



2. Model Building

Since we have one response variable and two independent variables we will have four models to choose from. Hence we build four models.

```
#----model1 with only b0
model1 <- function(){
  # model's likelihood
  for (i in 1:n){
    y[i] ~ dnorm(mu[i], tau)# stochastic componenent
    # link and linear predictor
    mu[i] <- beta0
  }
  # prior distributions
  tau ~ dgamma( 0.01, 0.01 )# stochastic componenent
  beta0 ~ dnorm( 0.0, 1.0E-4)# stochastic componenent
  # definition of sigma
  s2<-1/tau
  sigma <-sqrt(s2)
  # calculation of the sample variance
  for(ii in 1:n){
    c.time[ii]<-y[ii]-mean(y)
  }
  sy2 <- inprod(c.time[], c.time[] )/(n-1)
  # calculation of Bayesian version R squared
  R2B <- 1 - s2/sy2
  # Expected y for a typical delivery time
  expected.y <- beta0
  #
  # posterior probabilities of positive beta's
  p.beta0 <- step(beta0)
}

inits1 = inits_2 <- function(){
```

```
  list("tau" = 1, "beta0" = 1)}
n = NROW(y)
param.to.save = c("beta0","tau", "R2B", "sigma")
model1.data =list("n" = n,"y" = y)

fit.model1 <- jags(data = model1.data, inits = inits1, parameters.to.save
=param.to.save,
                   n.chains =3, n.iter = 2000, n.burnin = 1000,  n.thin = 1,
model.file = model1)
```

```
#---Building the model2 with assumption of b0 and b1
model2 <- function(){
  # model's likelihood
  for (i in 1:n){
    y[i] ~ dnorm(mu[i], tau )# stochastic componenent
    # link and linear predictor
    mu[i] <- beta0+inprod(beta[1:nPred] , X[i,1:nPred])
  }
  # prior distributions
  tau ~ dgamma( 0.01, 0.01 )# stochastic componenent
  beta0 ~ dnorm( 0.0, 1.0E-4)# stochastic componenent
  for(j in 1:nPred){
    beta[j] ~ dnorm( 0, 1.0E-4)# stochastic componenent
  }
  # definition of sigma
  s2<-1/tau
  sigma <-sqrt(s2)
  # calculation of the sample variance
  for(ii in 1:n){
    c.time[ii]<-y[ii]-mean(y)
  }
  sy2 <- inprod( c.time[], c.time[] )/(n-1)
  # calculation of Bayesian version R squared
  R2B <- 1 - s2/sy2
```

```
  # Expected y for a typical delivery time
  expected.y <- beta0 + beta[1] * mean(X[1:n, 1])
  #
  # posterior probabilities of positive beta's
  p.beta0 <- step(beta0)
  p.beta1 <- step(beta[1])
}



nPred = 1
inits1 = inits_2 <- function(){
  list("tau" = 1, "beta0" = 1, "beta" = rep(0, nPred))}
n = NROW(y)
param.to.save = c("beta0", "beta", "tau", "R2B", "sigma", "expected.y")
model1.data =list("n" = n, nPred = nPred, "y" = y, "X" = X)

fit.model2 <- jags(data = model1.data,inits = inits1, parame-
ters.to.save=param.to.save,n.chains =3, n.iter = 2000, n.burnin = 1000,  n.thin =
1, model.file = model2)

#--Now we build the model3 with variable Distance
model3 <- function(){
  # model's likelihood
  for (i in 1:n){
    y[i] ~ dnorm(mu[i], tau )# stochastic componenent
    # link and linear predictor
    mu[i] <- beta0+inprod(beta[1:nPred] , X[i,1:nPred])
  }
  # prior distributions
  tau ~ dgamma( 0.01, 0.01 )# stochastic componenent
  beta0 ~ dnorm( 0.0, 1.0E-4)# stochastic componenent
  for(j in 1:nPred){
    beta[j] ~ dnorm( 0, 1.0E-4)# stochastic componenent
  }
```

```
  # definition of sigma
  s2<-1/tau
  sigma <-sqrt(s2)
  # calculation of the sample variance
  for(ii in 1:n){
    c.time[ii]<-y[ii]-mean(y)
  }
  sy2 <- inprod( c.time[], c.time[] )/(n-1)
  # calculation of Bayesian version R squared
  R2B <- 1 - s2/sy2
  # Expected y for a typical delivery time
  expected.y <- beta0+ beta[1] * mean(X[1:n, 2])
  #
  # posterior probabilities of positive beta's
  p.beta0 <- step(beta0)
  p.beta2 <- step(beta[1])
}

nPred = 1
inits1 = inits_2 <- function(){
  list("tau" = 1, "beta0" = 1, "beta" = rep(0, nPred))}
n = NROW(y)
param.to.save = c("beta0", "beta", "tau", "R2B", "sigma", "expected.y")
model1.data =list("n" = n, nPred = nPred, "y" = y, "X" = X)

fit.model3 <- jags(data = model1.data, inits = inits1,            parame-
ters.to.save=param.to.save,n.chains =3, n.iter = 2000, n.burnin = 1000, n.thin =
1, model.file = model3)



#--Now we build model with all the features
model4 <- function(){
  # model's likelihood
  for (i in 1:n){
```

```
    y[i] ~ dnorm(mu[i], tau )# stochastic componenent
    # link and linear predictor
    mu[i] <- beta0+inprod(beta[1:nPred] , X[i,1:nPred])
  }
  # prior distributions
  tau ~ dgamma( 0.01, 0.01 )# stochastic componenent
  beta0 ~ dnorm( 0.0, 1.0E-4)# stochastic componenent
  for(j in 1:nPred){
    beta[j] ~ dnorm( 0, 1.0E-4)# stochastic componenent
  }
  # definition of sigma
  s2<-1/tau
  sigma <-sqrt(s2)
  # calculation of the sample variance
  for(ii in 1:n){
    c.time[ii]<-y[ii]-mean(y)
    }
  sy2 <- inprod( c.time[], c.time[] )/(n-1)
  # calculation of Bayesian version R squared
  R2B <- 1 - s2/sy2
  # Expected y for a typical delivery time
  expected.y <- beta0 + beta[1] * mean(X[1:n, 1]) + beta[2] * mean(X[1:n, 2])
  #
  # posterior probabilities of positive beta's
  p.beta0 <- step(beta0)
  p.beta1 <- step(beta[1])
  p.beta2 <- step(beta[2])
}

nPred = NCOL(X)
inits1 = inits_2 <- function(){
  list("tau" = 1, "beta0" = 1, "beta" = rep(0, nPred))}
n = NROW(y)
param.to.save = c("beta0", "beta", "tau", "R2B", "sigma", "expected.y")
model1.data =list("n" = n, nPred = nPred, "y" = y, "X" = X)
```

```
fit.model4 <- jags(data = model1.data, inits = inits1, parameters.to.save
=param.to.save,
                   n.chains =3, n.iter = 2000, n.burnin = 1000,  n.thin = 1,
model.file = model4)

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 25
##    Unobserved stochastic nodes: 4
##    Total graph size: 206
##
## Initializing model
```

3. Model Evaluation Tests

Now after creating the models we can easily computer the DIC model evaluation criteria. Natural way to compare models is to use criterion based on trade-off between the fit of the data to the model and the corresponding complexity of the model. In this project I compared four models.

Deviance Information Criterion, $DIC = $ $goodnessoffit$+ complexity

$DIC(m) = 2\overline{D(\theta_m, m)} - D(\overline{\theta}_m, m)$ $D(\overline{\theta}_m, m) + 2p_m$ $D(\theta_m, m) = -2logf(y \vee \theta_m, m)$ $\overline{D(\theta_m, m)}$ is deviance posterior mean. p_m is the number of effective parameters for model m $p_m = \overline{D(\theta_m, m)} - D(\overline{\theta}_m, m)$ and $\overline{\theta}$ is the posterior mean of parameters involved in model m.

For our case JAGS Automatically calculates DIC. The smaller the DIC the better model as supported by theory.

Here below I compare models using DIC and I also considered the calculated Bayesian version of R-squared, which I expected the better supported model to have a high score.

```
#Now we can compare our models via DIC(Deviance Information Criteria)
dic.samples(fit.model1$model,n.iter =2e3)

## Mean deviance:  209.1
## penalty 2.22
## Penalized deviance: 211.4

fit.model1$BUGSoutput$mean$R2B

## [1] -0.09410882

#model 2
dic.samples(fit.model2$model,n.iter =2e3)

## Mean deviance:  143.5
## penalty 3.21
## Penalized deviance: 146.8

fit.model2$BUGSoutput$mean$R2B

## [1] 0.9171794

#model 3
dic.samples(fit.model3$model,n.iter =2e3)
```

```
## Mean deviance:  143.5
## penalty 3.283
## Penalized deviance: 146.8

fit.model3$BUGSoutput$mean$R2B

## [1] 0.9168237

#model4
dic.samples(fit.model4$model,n.iter =2e3)

## Mean deviance:  131.2
## penalty 4.385
## Penalized deviance: 135.5

fit.model4$BUGSoutput$mean$R2B

## [1] 0.9486455
```

As can be seen that model4 has the lowest DIC hence according to theory is our best model out of all

4. MCMC Analysis and Convergence Diagnostics

Checking model convergence, I have decided to use Gelman-Rubin diagnostics plots and also Heildel diagnostics to see if each parameter has passed the invariant distribution.

Gelman Diagnostics are calculated as below: 1. Compute m independent Markov chains 2. compare the variance of each chain to pooled variance Therefore, provides an estimate of how much variance could be reduced by running chains longer. As for Heildel diagnostics we can see that all the parameters have passed the test for invariant distribution. This indicates that there is actually convergence.

```
mcmc.model4 = as.mcmc(fit.model4)
heidel.diag(mcmc.model4)
```

```
## [[1]]
##
##            Stationarity start     p-value
##            test         iteration
## beta0      passed         1       0.5612
## beta[1]    passed         1       0.2247
## beta[2]    passed         1       0.3470
## deviance   passed         1       0.0842
## expected.y passed         1       0.9612
## R2B        passed       101       0.6944
## sigma      passed       101       0.7150
## tau        passed         1       0.5568
##
##            Halfwidth Mean     Halfwidth
##            test
## beta0      passed      2.3639 0.079758
## beta[1]    passed      1.6161 0.010807
## beta[2]    passed      0.0143 0.000219
## deviance   passed    131.2906 0.299271
## expected.y passed     22.3701 0.043500
## R2B        passed      0.9512 0.001187
## sigma      passed      3.3834 0.040126
```

```
## tau         passed      0.0939 0.002054
##
## [[2]]
##
##             Stationarity start     p-value
##             test         iteration
## beta0       passed         1       0.780
## beta[1]     passed         1       0.375
## beta[2]     passed         1       0.697
## deviance    passed       101       0.318
## expected.y  passed         1       0.807
## R2B         passed       101       0.983
## sigma       passed       101       0.975
## tau         passed         1       0.508
##
##             Halfwidth Mean     Halfwidth
##             test
## beta0       passed      2.3086 0.074283
## beta[1]     passed      1.6222 0.011418
## beta[2]     passed      0.0143 0.000229
## deviance    passed    131.0473 0.239184
## expected.y  passed     22.3595 0.042174
## R2B         passed      0.9525 0.001182
## sigma       passed      3.3421 0.040037
## tau         passed      0.0954 0.002079
##
## [[3]]
##
##             Stationarity start     p-value
##             test         iteration
## beta0       passed         1       0.251
## beta[1]     passed         1       0.661
## beta[2]     passed         1       0.166
## deviance    passed       101       0.730
## expected.y  passed         1       0.882
```

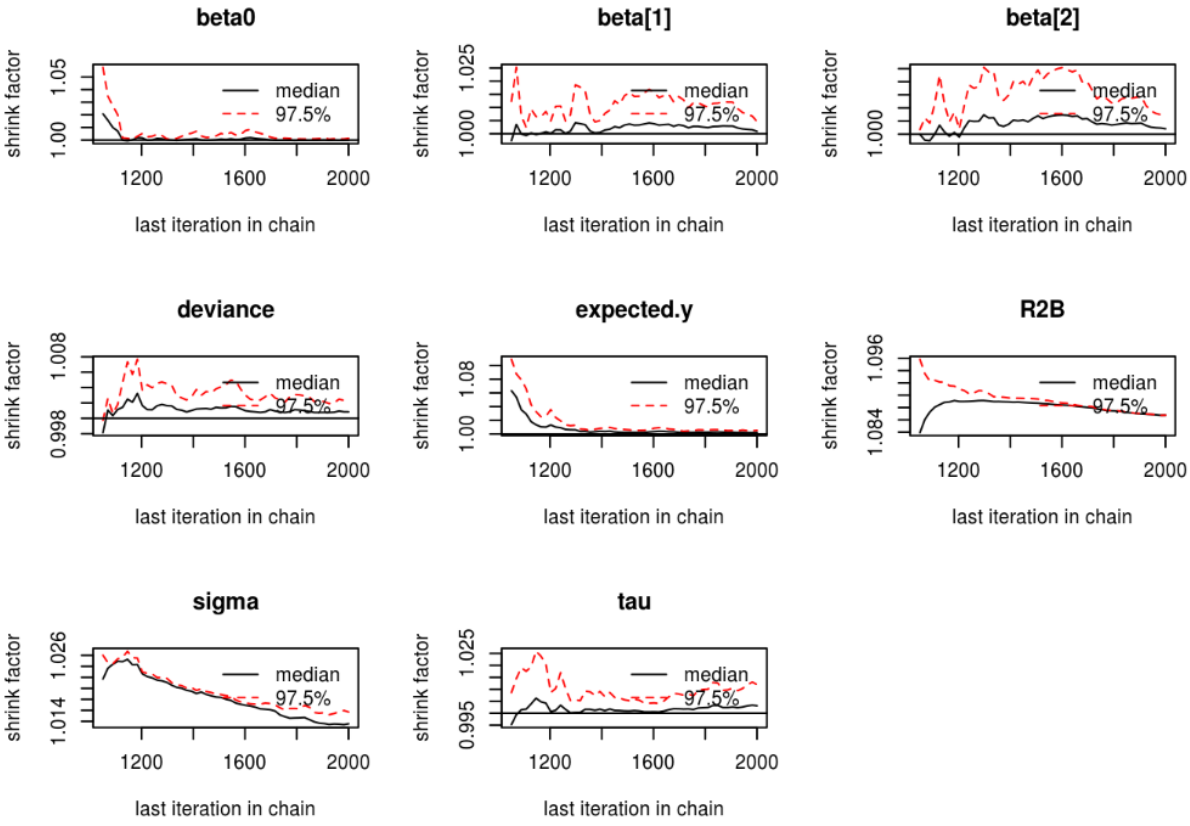
```
## R2B         passed       101       0.602
## sigma       passed       101       0.521
## tau         passed         1       0.566
##
##             Halfwidth Mean     Halfwidth
##             test
## beta0       passed      2.2562 0.069393
## beta[1]     passed      1.6316 0.012014
## beta[2]     passed      0.0141 0.000245
## deviance    passed    131.1228 0.201281
## expected.y passed     22.3375 0.042129
## R2B         passed      0.9518 0.001260
## sigma       passed      3.3619 0.042187
## tau         passed      0.0951 0.002338
```

```
summary(mcmc.model4)
```

```
##
## Iterations = 1001:2000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 1000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                 Mean       SD  Naive SE Time-series SE
## beta0        2.30957 1.195947 2.183e-02      2.197e-02
## beta[1]      1.62329 0.187561 3.424e-03      3.365e-03
## beta[2]      0.01424 0.003845 7.019e-05      6.816e-05
## deviance   131.25397 4.100069 7.486e-02      9.819e-02
## expected.y  22.35568 0.687314 1.255e-02      1.255e-02
## R2B          0.94865 0.091794 1.676e-03      1.841e-03
## sigma        3.39463 0.924324 1.688e-02      2.064e-02
## tau          0.09478 0.029511 5.388e-04      6.365e-04
```

```
## 2. Quantiles for each variable:
##
##                  2.5%       25%       50%       75%     97.5%
## beta0       -0.023036   1.53742   2.32377   3.0962   4.60671
## beta[1]      1.280258   1.50582   1.62332   1.7401   1.98210
## beta[2]      0.007018   0.01191   0.01429   0.0166   0.02153
## deviance   127.345276 128.88379 130.41382 132.6588 138.99058
## expected.y  20.997513  21.90884  22.36515  22.8000  23.68539
## R2B          0.909999   0.94409   0.95482   0.9631   0.97398
## sigma        2.504359   2.98341   3.29999   3.6708   4.65749
## tau          0.046099   0.07421   0.09183   0.1124   0.15944
```

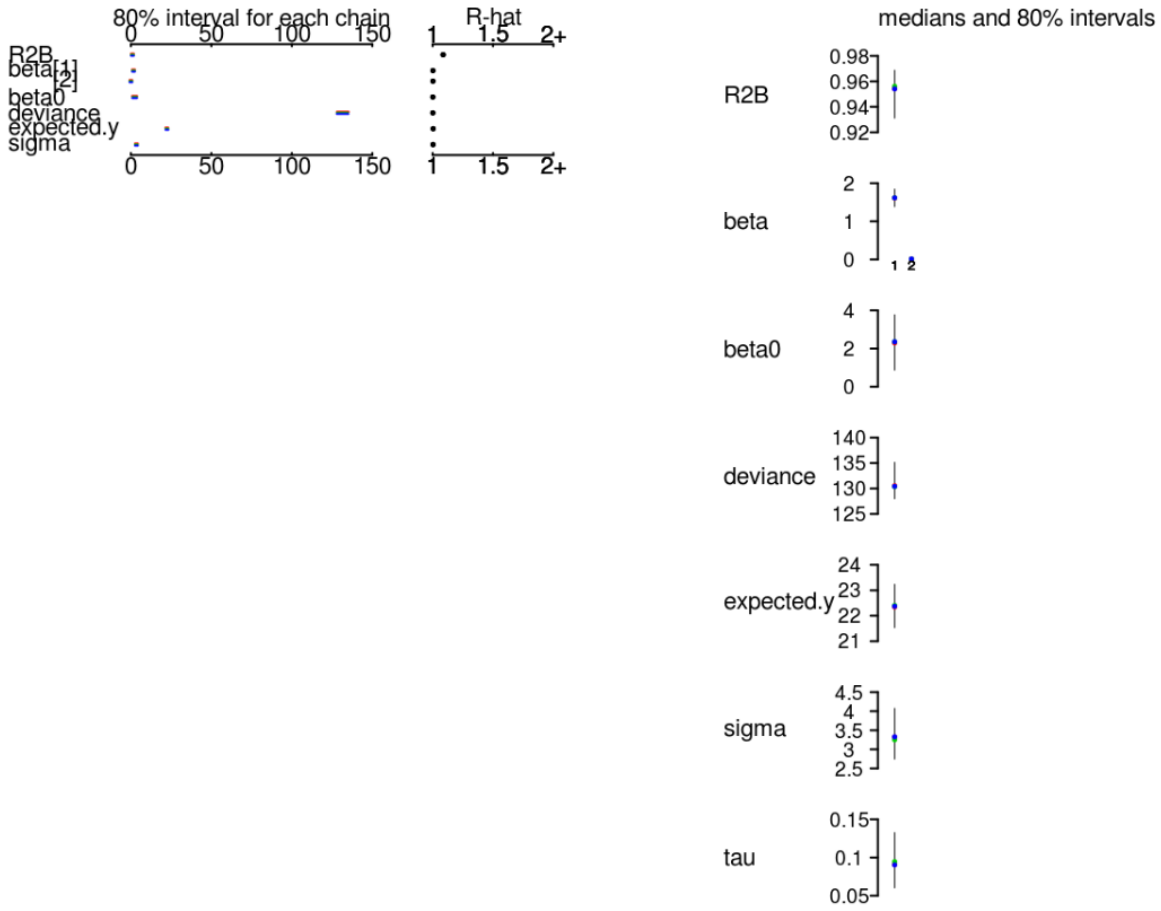
```
gelman.plot(mcmc.model4)
```



The gelman plot shows you the development of the scale-reduction over time (chain steps), which is useful to see whether a low chain reduction is also stable (sometimes, the factors go down and then up again, as can be seen. For this analysis I have already discarded values before the period considered to be burn-in

Analysis of the model via posterior values

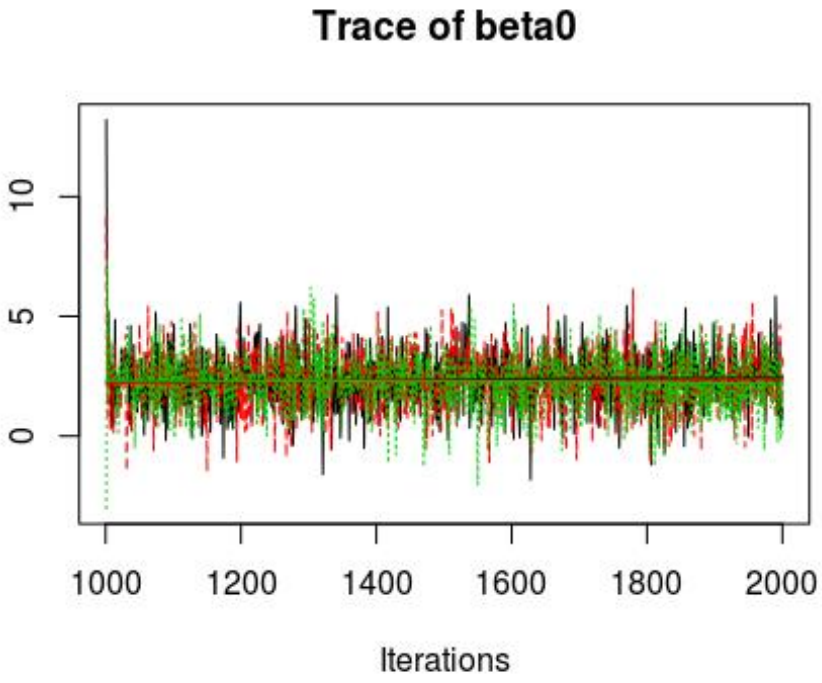
```
plot(fit.model4)
```

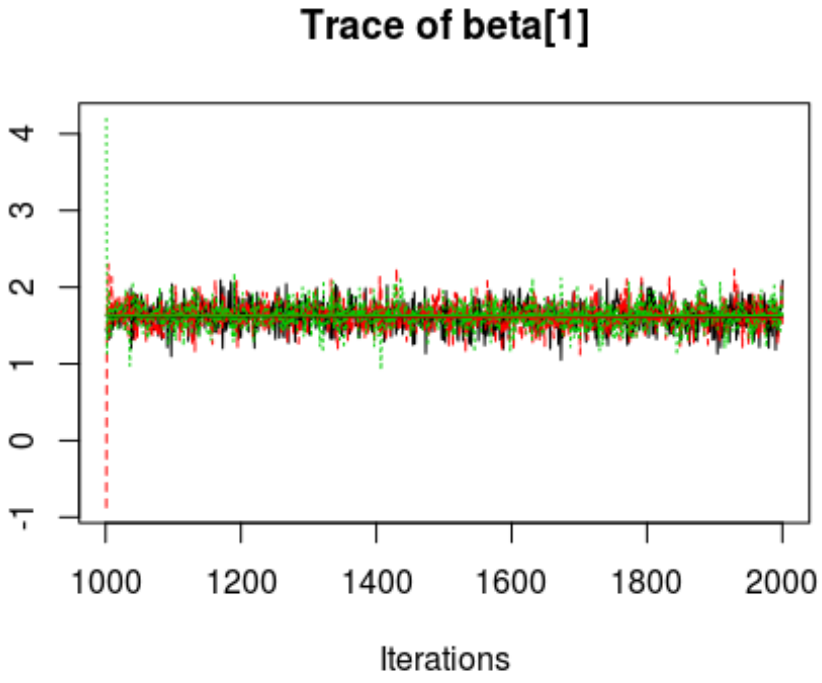


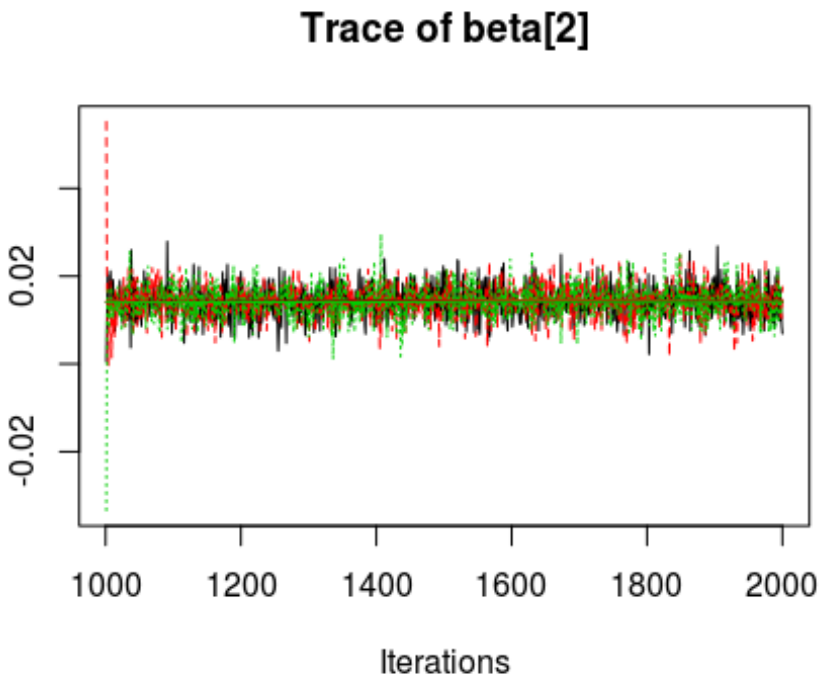
Analysis of the plots above:

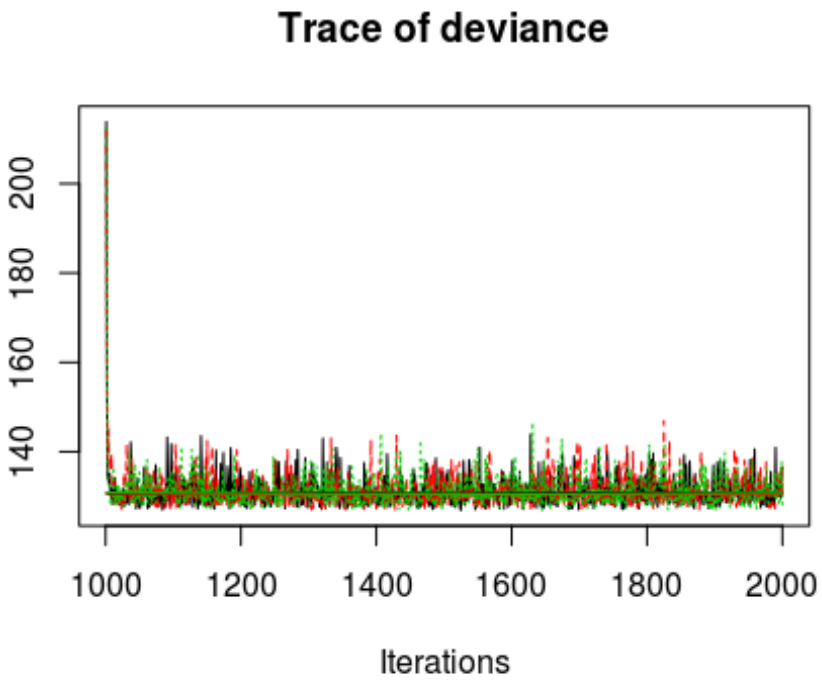
statistic for each parameter provides a measure of sampling efficiency/effectiveness. Ideally, all values should be less than 1.05. If there are values of 1.05 or greater it suggests that the sampler was not very efficient or effective. Not only does this mean that the sampler was potential slower than it could have been, more importantly, it could indicate that the sampler spent time sampling in a region of the likelihood that is less informative. Such a situation can arise from either a misspecified model or overly vague priors that permit sampling in otherwise nonsense parameter space. As can be seen in the plots for R-hat all parameter values are less than 1.05 which is a good thing.

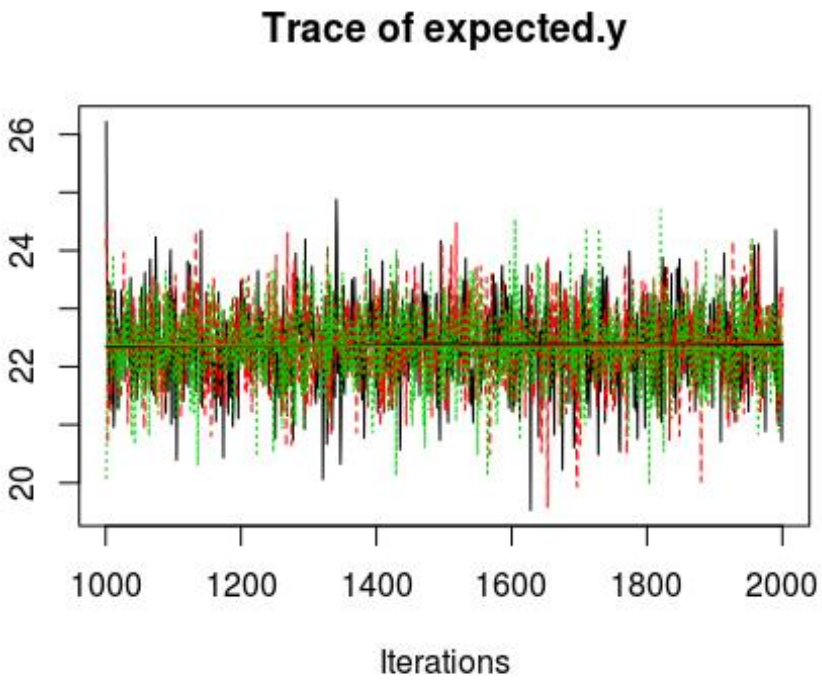
```r
#plotting of history of parameters
mcmc.model4 <- as.mcmc(fit.model4)
traceplot(mcmc.model4)
```

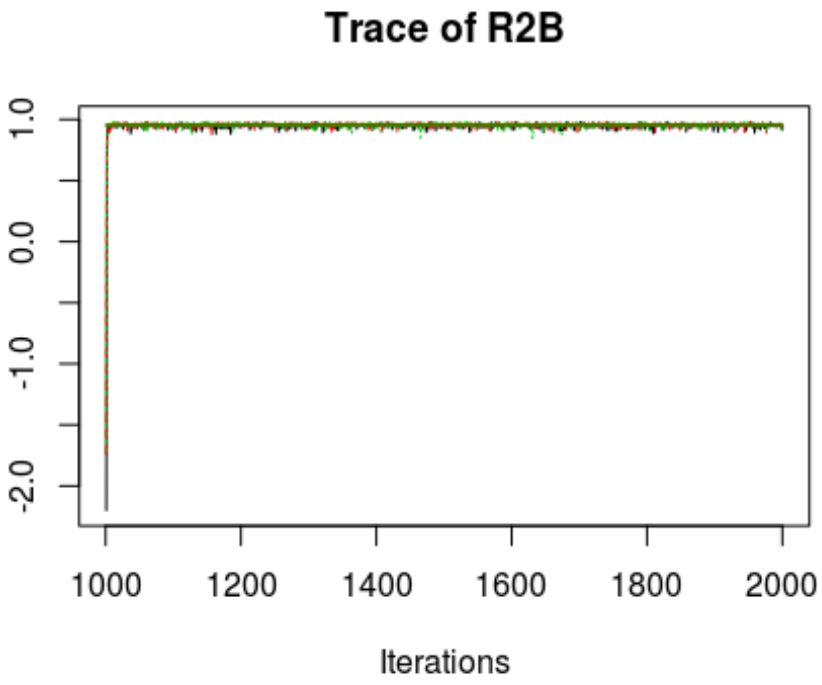

Trace of beta0
10
5
0
1000
1200
1400
1600
1800
2000
Iterations

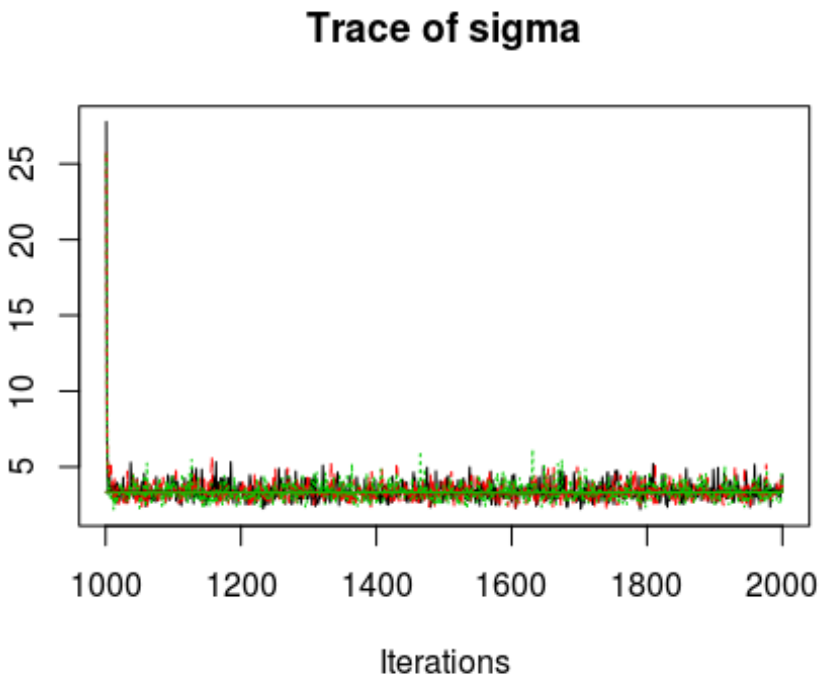

Trace of beta[1]
4
3
2
1
0
-1
1000
1200
1400
1600
1800
2000
Iterations

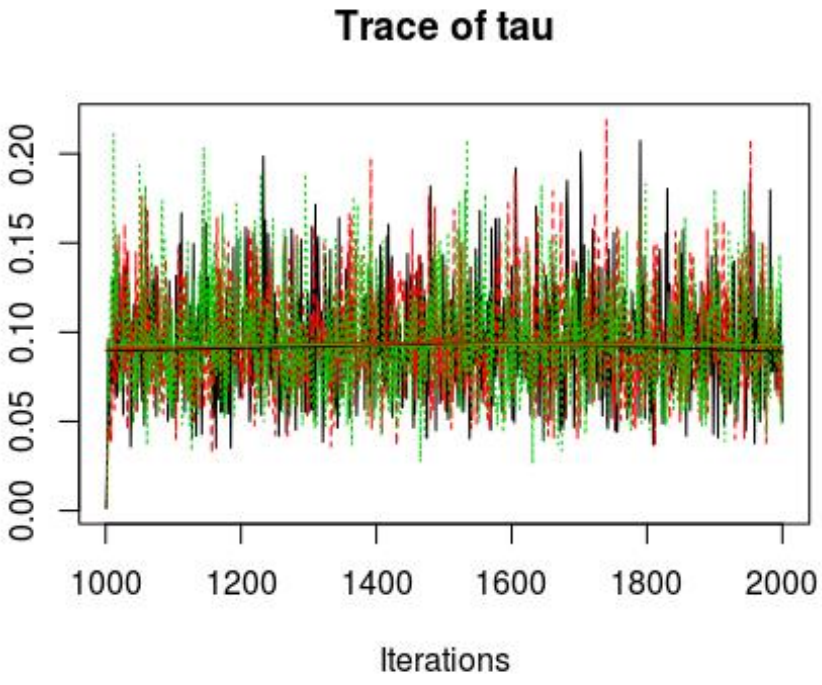

Trace of beta[2]
0.02
-0.02
1000
1200
1400
1600
1800
2000
Iterations


Trace of deviance
200
180
160
140
1000
1200
1400
1600
1800
2000
Iterations


Trace of expected.y
20
22
24
26
1000
1200
1400
1600
1800
2000
Iterations


Trace of R2B
-2.0
-1.0
0.0
1.0
1000
1200
1400
1600
1800
2000
Iterations

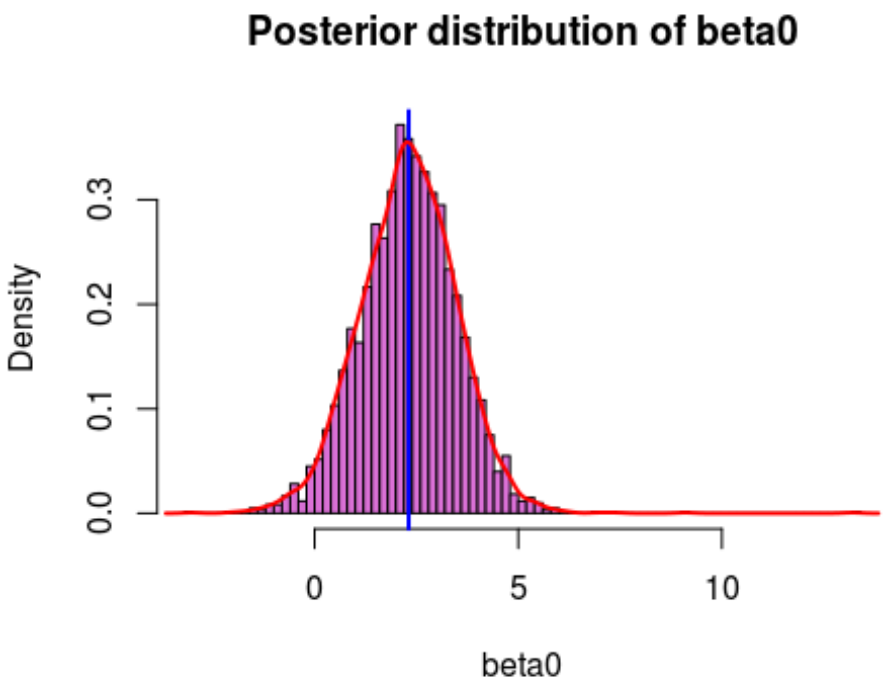

Trace of sigma
5
10
15
20
25
1000
1200
1400
1600
1800
2000
Iterations


Trace of tau
0.00
0.05
0.10
0.15
0.20
1000
1200
1400
1600
1800
2000
Iterations

Plots of histograms of these posterior values of interest. These are density plots after a burn-in of 1000

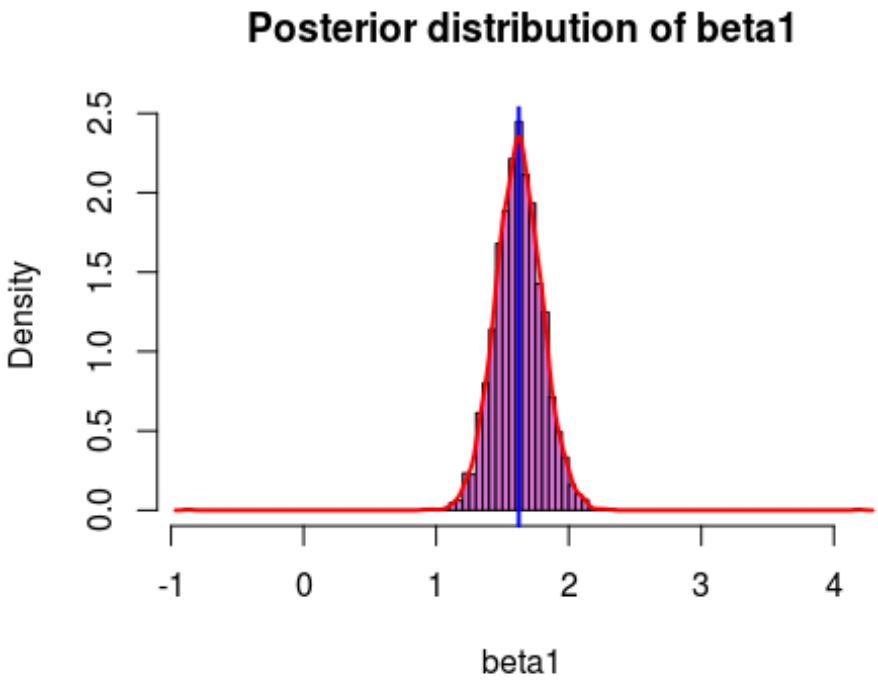
```
output <-fit.model4$BUGSoutput$sims.matrix
beta0 <- output[,"beta0"]
beta1 <- output[,"beta[1]"]
beta2 <- output[,"beta[2]"]
tau <- output[,"tau"]
sigma <- output[,"sigma"]
expected.y <- output[,"expected.y"]


hist(beta0,col='orchid',breaks = 80, prob = T, main = "Posterior distribution of
beta0")
abline(v = mean(beta0), col = "blue", lwd = 2)
lines(density(beta0),col='red',lwd=2, lty = c(1, 3))
```

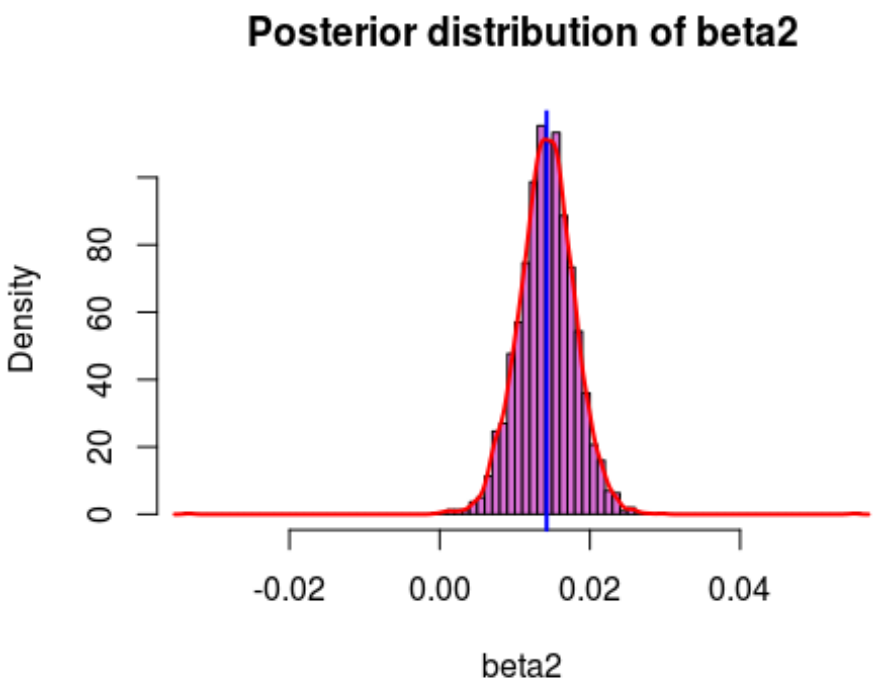



```
hist(beta1,col='orchid',breaks = 80, prob = T, main = "Posterior distribution of
beta1")
```

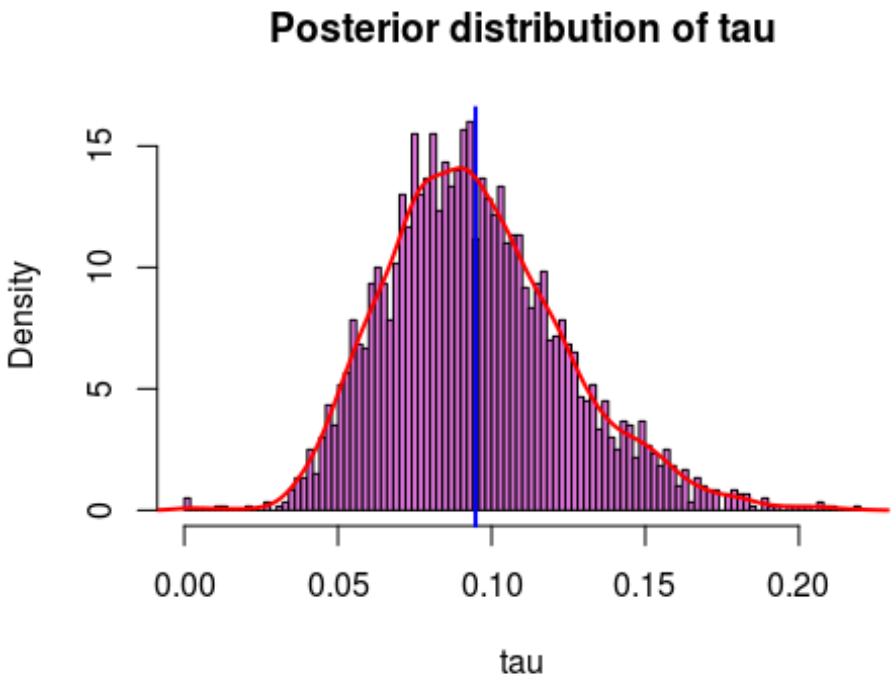
```
abline(v = mean(beta1), col = "blue", lwd = 2)
lines(density(beta1),col='red',lwd=2, lty = c(1, 3))
```



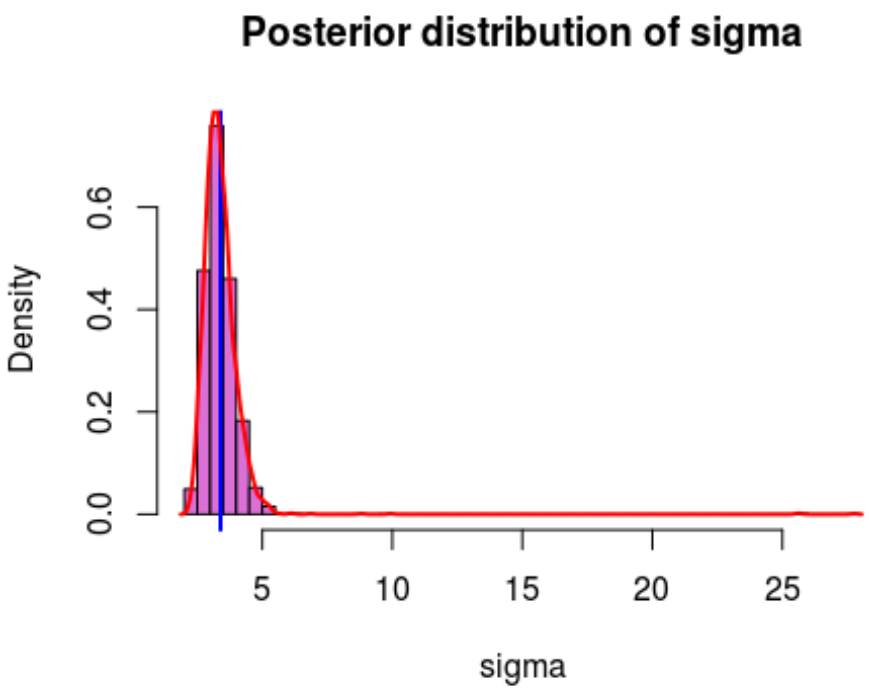

```
hist(beta2,col='orchid',breaks = 80, prob= T, main = "Posterior distribution of
beta2")
abline(v = mean(beta2), col = "blue", lwd = 2)
lines(density(beta2),col='red',lwd=2, lty = c(1, 3))
```



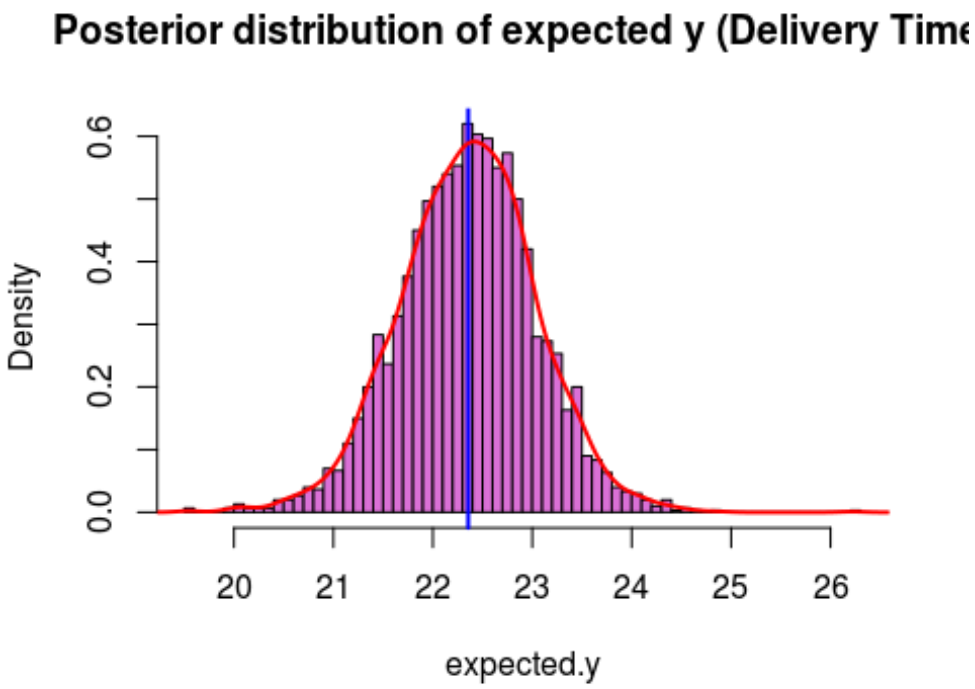

```
hist(tau,col='orchid',breaks = 80, prob = T, main = "Posterior distribution of
tau")
abline(v = mean(tau), col = "blue", lwd = 2)
lines(density(tau),col='red',lwd=2)
```




```
hist(sigma,col='orchid',breaks = 80, prob = T, main = "Posterior distribution of
sigma")
abline(v = mean(sigma), col = "blue", lwd = 2)
lines(density(sigma),col='red',lwd=2)
```




```
hist(expected.y,col='orchid',breaks = 80, prob = T, main = "Posterior distribution
of expected y (Delivery Time)")
abline(v = mean(expected.y), col = "blue", lwd = 2)
lines(density(expected.y),col='red',lwd=2)
```


Posterior distribution of expected y (Delivery Time
Density
0.0
0.2
0.4
0.6
20
21
22
23
24
25
26
expected.y

5. Comparison of Chosen Model with Frequentist Approach

Now we can compare the Frequentist approach and the Bayesian parameter estimation for our model:

```
#Bayesian estimated coefficients
beta0 <- fit.model4$BUGSoutput$mean$beta0
beta <-  fit.model4$BUGSoutput$mean$beta
c("b0" = beta0, "b" = beta)

##         b0         b1         b2
## 2.30956525 1.62328774 0.01423504

bayes_coef =c("b0" = beta0, "b" = beta)
#Now we fit the linear model and estimate the parameters
fit <- lm(my.data$Time~., data =my.data[c(FALSE, TRUE, TRUE)])
fit$coefficients

## (Intercept)       Cases    Distance
##  2.34123115  1.61590721  0.01438483
```

As can be seen from the estimated coefficients of regression in both frequentist approach and Bayesian approach. The values are almost exact.

6. References

1. Ioannis Ntzoufras, (2010) “Bayesian Modeling Using winBUGS”
2. Petri Koistinen, (2010), “Monte Carlo Methods, with an emphasis on Bayesian computation”.
3. *http://webpages.math.luc.edu/~ebalderama/myfiles/modelchecking101_pres.pdf*