Bayesian Analysis

We realize that the output of `MCMClogit` function used for fitting a Bayesian logistic regression are the posterior samples of the model parameters with MCMC class. Therefore, we will not use the DIC for comparing the fitted models in this paradigm since it is difficult with our output. Instead, we are going to use the insights that we got in the Frequentist analysis. Hence, we will use the predictors of the best frequentist model **freq_glm_step** based on their AIC.

Non-informative prior

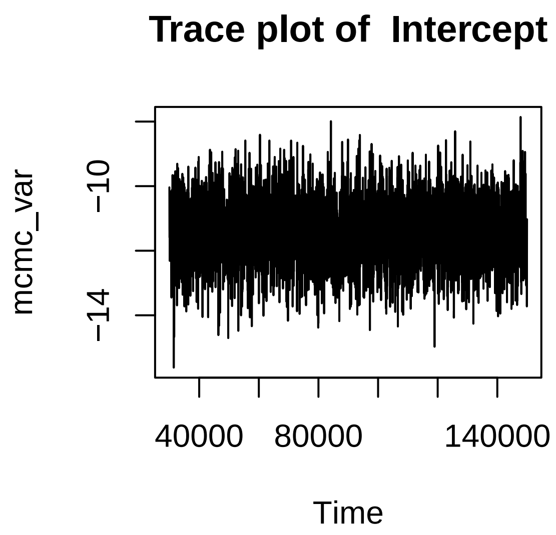
Firstly we set 120000 iterations for the sampling. We have also decided to use a burning of 30000 iterations to give the Markov Chain time to reach its stationary distribution. Additionally, we use a thinning of 30 to avoid autocorrelation between the successive samples in the MCMC resultant sample and we will obtain samples of 4000 values.

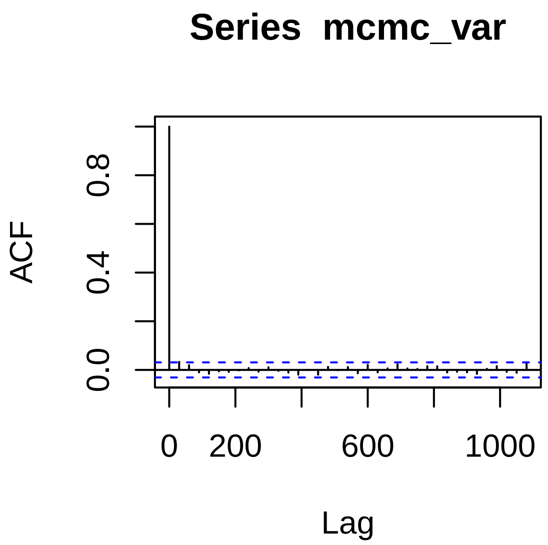
We begin assigning non-informative priors for the model parameters by fixing priori Normal distributions with zero mean and very small precision. The precision is referred the inverse of the variance.

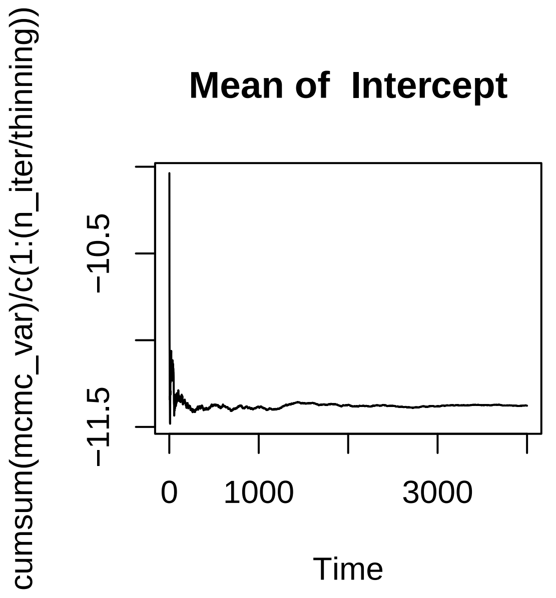
```r
n_iter <- 120000
thinning <- 30
burning <- 30000
#No informative prior
glm.bayes <- MCMClogit(Outcome ~ Pregnancies + Glucose + Insulin + BMI +  DiabetesPedigreeFunction,
                       data=diabetes_train,
                       thin=thinning,
                       mcmc=n_iter,
                       burnin=burning,
                       b0 = 0,
                       B0 = 0.00001)
```

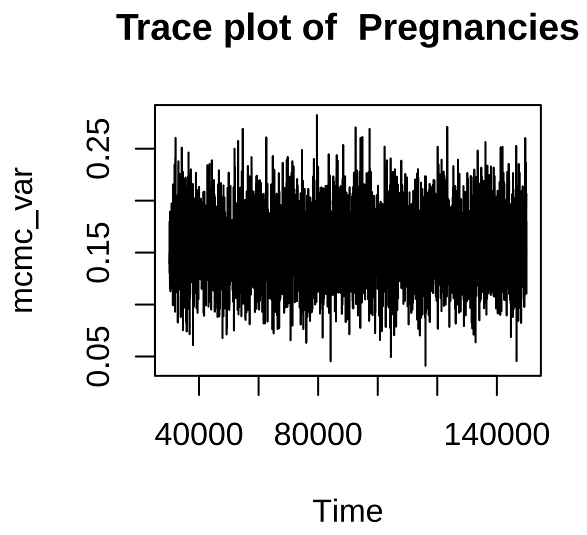
In order to check the performance of the MCMC chains of the posterior draws, we check the trace plot and the ACF for each parameter sample. We also plot the accumulated mean of each parameter to show the evolution of each mean value.

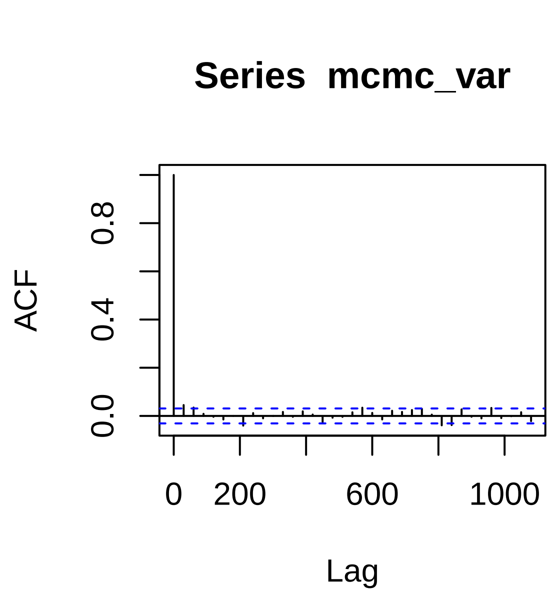
```r
par(mfrow = c(1,2))

var <- c('Intercept','Pregnancies','Glucose','Insulin','BMI','DiabetesPedigreeFunction')
mcmc_var <- rep(0,n_iter/thinning)
for (i in 1:6){
mcmc_var <- glm.bayes[,i]
ts.plot(mcmc_var,main=paste("Trace plot of ",var[i]))
acf(mcmc_var)
ts.plot(cumsum(mcmc_var)/c(1:(n_iter/thinning)),main=paste("Mean of ",var[i]))
}
```


Trace plot of Intercept
mcmc_var
−10
−14
40000
80000
140000
Time

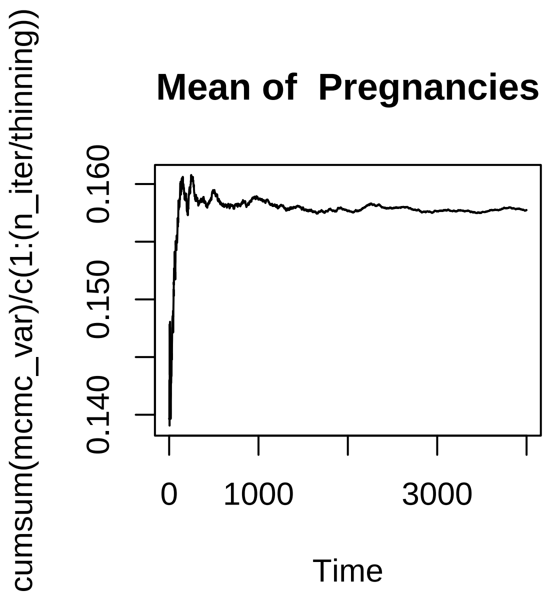

Series mcmc_var
ACF
0.0
0.4
0.8
0
200
600
1000
Lag


Mean of Intercept
cumsum(mcmc_var)/c(1:(n_iter/thinning))
−10.5
−11.5
0
1000
3000
Time

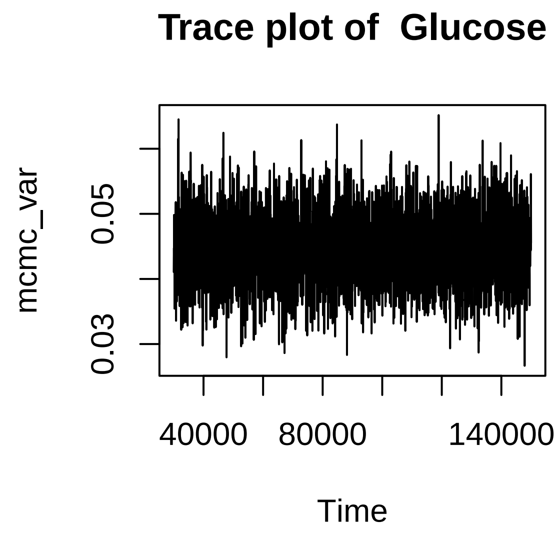

Trace plot of Pregnancies
mcmc_var
0.05
0.15
0.25
40000
80000
140000
Time



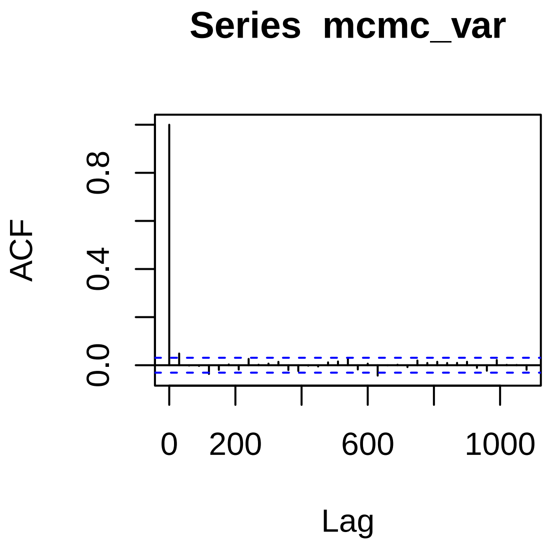
Series mcmc_var
ACF
0.0
0.4
0.8
0
200
600
1000
Lag



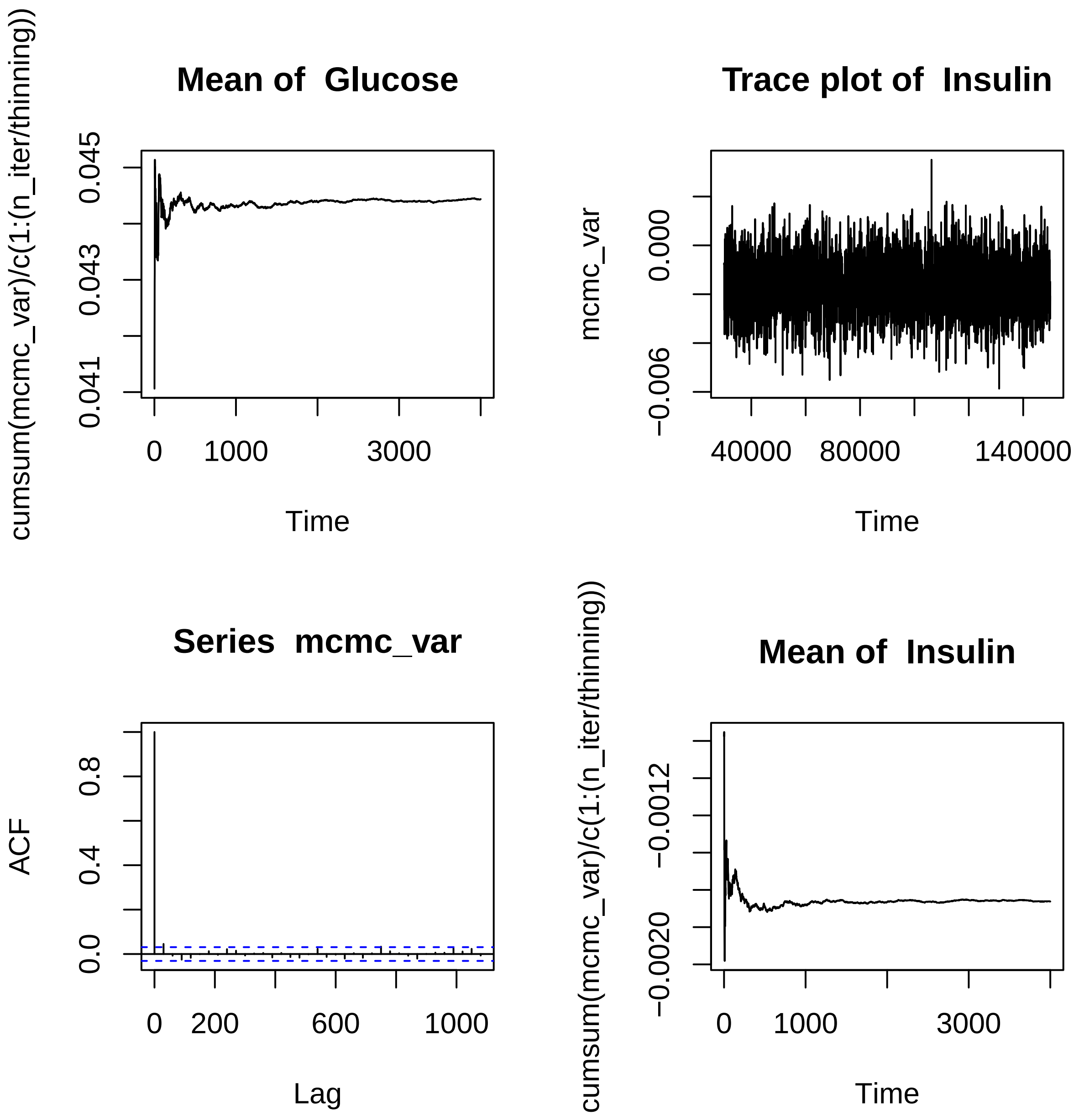
Mean of Pregnancies
cumsum(mcmc_var)/c(1:(n_iter/thinning))
0.140
0.150
0.160
0
1000
3000
Time

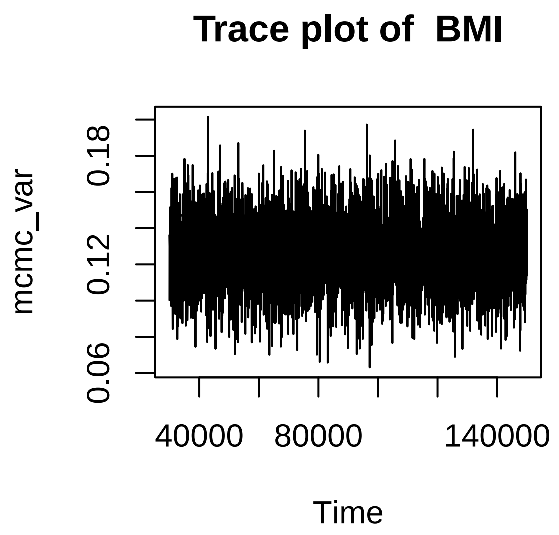


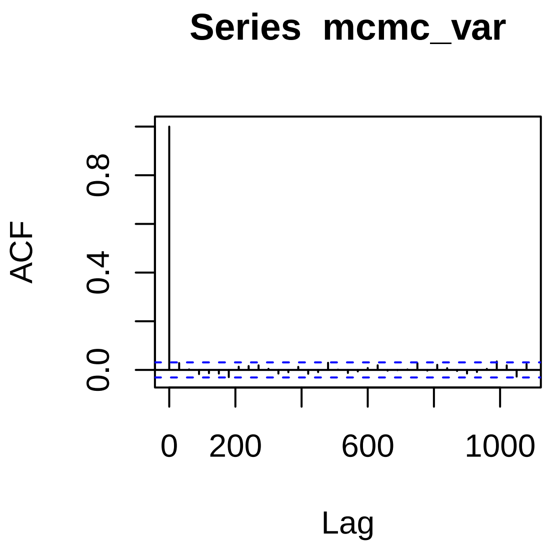
Trace plot of Glucose
mcmc_var
0.03
0.05
40000
80000
140000
Time

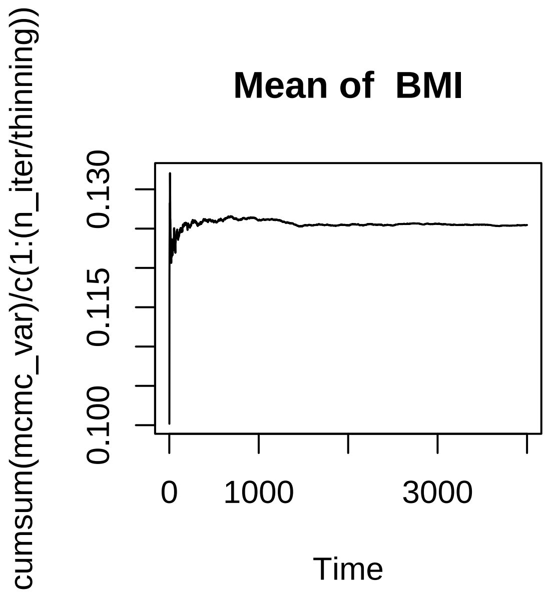


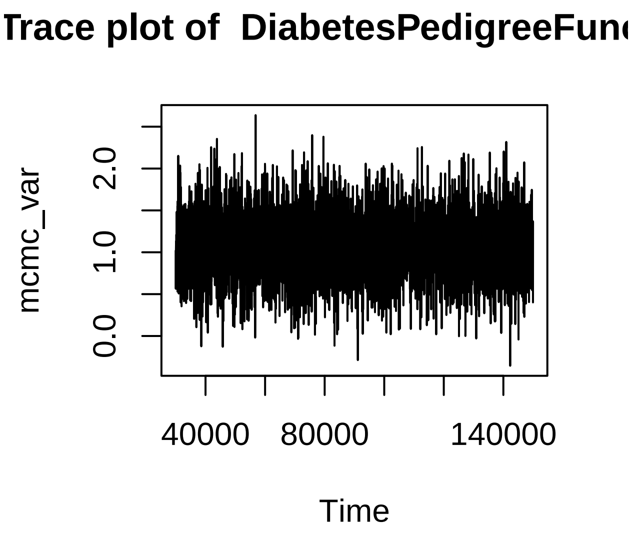
Series mcmc_var
ACF
0.0
0.4
0.8
0
200
600
1000
Lag

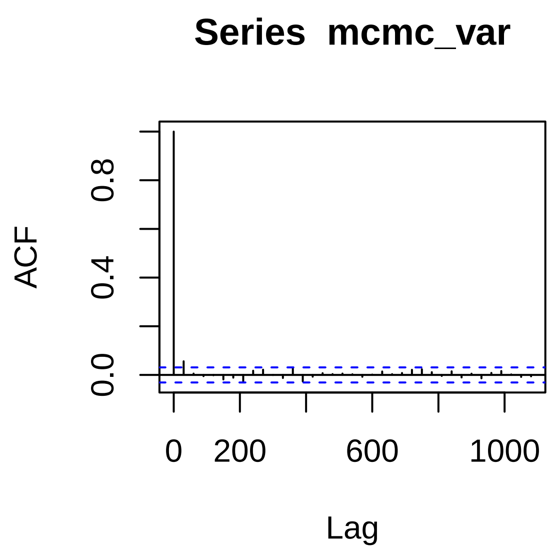

Mean of Glucose
cumsum(mcmc_var)/c(1:(n_iter/thinning))
0.041
0.043
0.045
0
1000
3000
Time
Trace plot of Insulin
mcmc_var
−0.006
0.000
40000
80000
140000
Time
Series mcmc_var
ACF
0.0
0.4
0.8
0
200
600
1000
Lag
Mean of Insulin
cumsum(mcmc_var)/c(1:(n_iter/thinning))
−0.0020
−0.0012
0
1000
3000
Time

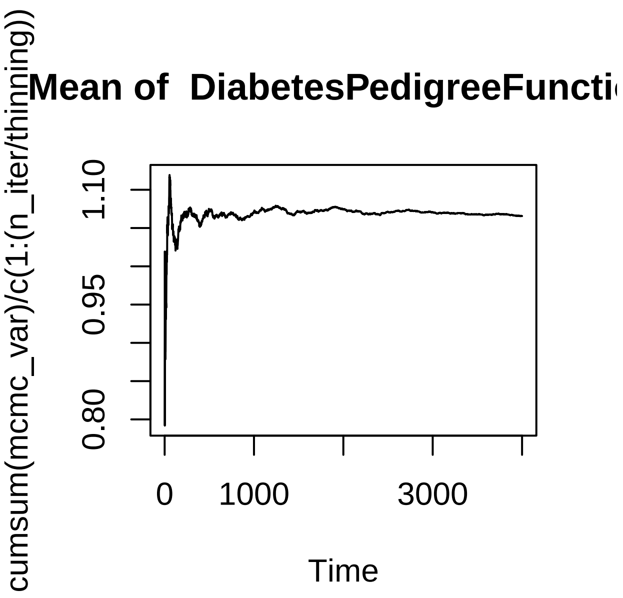

Trace plot of BMI
mcmc_var
0.06
0.12
0.18
40000
80000
140000
Time


Series mcmc_var
ACF
0.0
0.4
0.8
0
200
600
1000
Lag


Mean of BMI
cumsum(mcmc_var)/c(1:(n_iter/thinning))
0.100
0.115
0.130
0
1000
3000
Time


Trace plot of DiabetesPedigreeFunc
mcmc_var
0.0
1.0
2.0
40000
80000
140000
Time



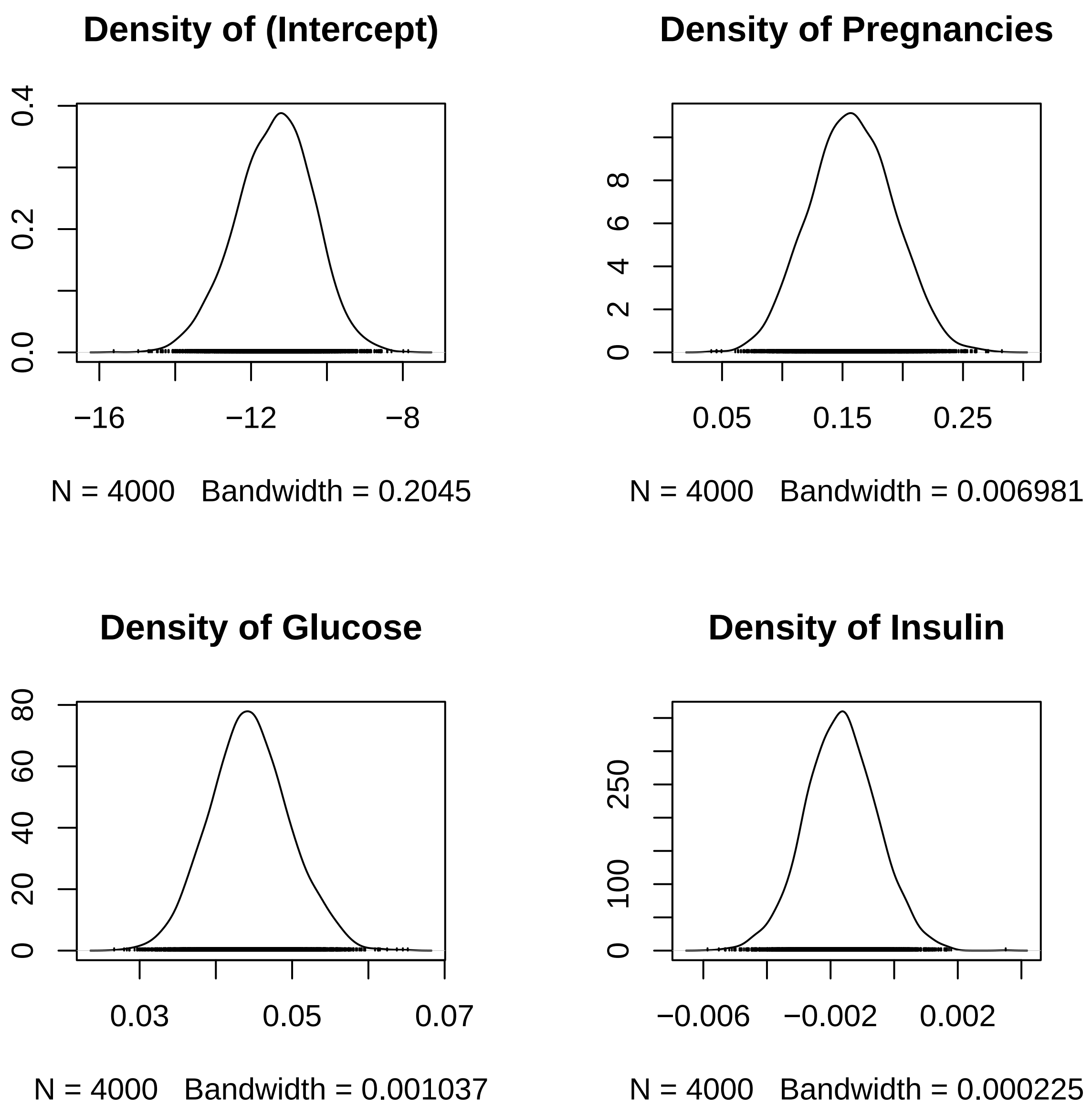


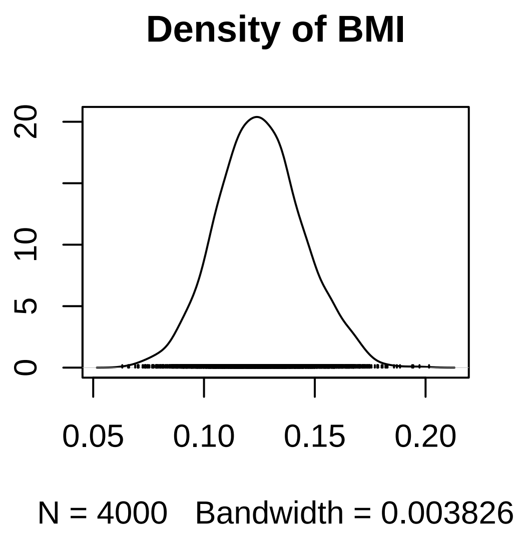
We can observe in the former graphs that for each parameter the trace plots resemble to be white noise and the ACF values for almost all the lags are no significant because they lie between the blue dashed lines. We also notice that after some iterations the mean values stabilize.

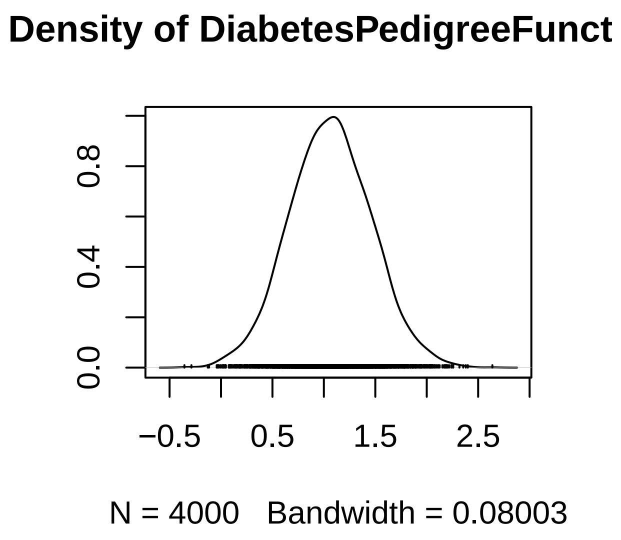
```
par(mfrow = c(1,2))
summary(glm.bayes)
```

```
##
## Iterations = 30001:149971
## Thinning interval = 30
## Number of chains = 1
## Sample size per chain = 4000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                                Mean       SD  Naive SE Time-series SE
## (Intercept)              -11.377829 1.013258 1.602e-02      1.657e-02
## Pregnancies                0.157737 0.034598 5.470e-04      5.915e-04
## Glucose                    0.044438 0.005229 8.268e-05      8.682e-05
## Insulin                   -0.001662 0.001128 1.783e-05      1.866e-05
## BMI                        0.125459 0.019434 3.073e-04      3.161e-04
## DiabetesPedigreeFunction   1.065640 0.396619 6.271e-03      6.634e-03
##
## 2. Quantiles for each variable:
##
##                               2.5%        25%        50%        75%      97.5%
## (Intercept)              -13.43604 -12.051885 -11.332948 -10.685450 -9.4568110
## Pregnancies                0.09205   0.133977   0.157497   0.181178  0.2253678
## Glucose                    0.03443   0.040928   0.044348   0.047818  0.0551530
## Insulin                   -0.00388  -0.002413  -0.001662  -0.000919  0.0005612
## BMI                        0.08836   0.112290   0.124943   0.137693  0.1654697
## DiabetesPedigreeFunction   0.29426   0.797207   1.063953   1.331737  1.8605678
```

```
densplot(glm.bayes)
```







Once we have the posteriory density functions for every parameter, we can make punctual predictions using the mean of each random variable and computing the logistic function:

$$y_{prediction} = \frac{1}{1 + e^{-X\beta}}$$

We set the threshold to an arbitrary number of *0.50*, which decides the corresponding class for a given prediction. This value can also be optimized when looking for a more balanced model, but we will not focus on that subject.

```
X_new <- cbind(1,X_test[,c(1,2,5,6,7)]) # Order the design matrix
beta_means <- summary(glm.bayes)$statistics[,1]

# Vector multiplication
z = as.matrix(X_new) %*% as.matrix(beta_means)
y_pred <- 1/(1+exp(-z))

# Hit matrix and hit ratio (Ridge model 1-se)
H <- table(y_pred> 0.5, y_test == 1)
H
```

```
##
##         FALSE TRUE
##   FALSE   124   48
##   TRUE     18   41
```

```
tp = H[2,2]
tn = H[1,1]
fp = H[2,1]
fn = H[1,2]
total = tp+tn+fp+fn

acc = (tp+tn)/total
rec = (tp)/(tp+fp)
pre = (tp)/(tp+fn)
F1 = 2*rec*pre/(pre+rec)
```

```
print(paste('Accuracy: ',acc ))
```

```
## [1] "Accuracy:  0.714285714285714"
```

```
print(paste('Recall: ', rec))
```

```
## [1] "Recall:  0.694915254237288"
```

```
print(paste('Precision: ',pre))
```

```
## [1] "Precision:  0.460674157303371"
```

```
print(paste('F1-score: ',F1 ))
```

```
## [1] "F1-score:  0.554054054054054"
```

```
D = freq_glm$deviance
D0 = freq_glm$null.deviance

print(paste('R2: ', 1-D/D0 ))
```

```
## [1] "R2:  0.327058067272475"
```

The results of this model are really similar from the obtained before, both in predictions and in model parameters (in terms of means). This is reasonable, as we have not defined any prior density, and the values captured by the model are just the data. However, the Bayesian model offers the flexibility of giving a density as output, so we can estimate the prediction using means, median and even quantiles.

Informative prior

To detail the prior density function, we will use some medical insights to try to deduce *a priori* the parameters of the model. For constructing the priors, we have taken the following steps:

- For variables which are positively correlated with diabetes, a flat exponential is set. This seeks to assign prior probabilities only to the positive values. Those variables are: *Pregrancies, Glucose, BMI, Diabetes predigree.*
- For variables with negative correlation, we set the exponential of $(-x_j)$, which assign prior probabilities to negative values only. Those variables are: *Insulin.*
- For variables with no prior, we set the typical Gaussian with high variance.

The joint density function is set as the multiplication of all the corresponding densities, this assumes independence (which is not entirely true, but works as prior).

The same parameters as the previous case had been used for the MCM chain.

```
n_iter <- 120000
thinning <- 30
burning <- 30000


logprior <- function(beta){

         dnorm(beta[1],sd=500)* #The intercept is neutral, we do not have prior
         dexp(beta[2],rate=.1)*
         dexp(-beta[3],rate=.1)*
         dexp(beta[4],rate=.1)*
         dexp(beta[5],rate=.1)

}
```

```
#Informative prior
glm.bayes2 <- MCMClogit(Outcome ~ Pregnancies + Glucose + Insulin + BMI +  DiabetesPedigreeFunction,
                      data=diabetes_train,
                      thin=thinning,
                      mcmc=n_iter,
                      burnin=burning,
                      user.prior.density=logprior)
```

```
##
##
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
## The Metropolis acceptance rate was 0.23145
## @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

```
summary(glm.bayes2)
```

```
##
## Iterations = 30001:149971
## Thinning interval = 30
## Number of chains = 1
## Sample size per chain = 4000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##                                Mean       SD  Naive SE Time-series SE
## (Intercept)              -11.386075 1.015999 1.606e-02      1.651e-02
## Pregnancies                0.157352 0.034671 5.482e-04      5.939e-04
## Glucose                    0.044462 0.005203 8.226e-05      8.128e-05
## Insulin                   -0.001666 0.001127 1.782e-05      1.841e-05
## BMI                        0.125604 0.019398 3.067e-04      3.168e-04
## DiabetesPedigreeFunction   1.068718 0.398587 6.302e-03      6.722e-03
##
## 2. Quantiles for each variable:
##
##                                2.5%        25%        50%        75%      97.5%
## (Intercept)              -13.442346 -12.049014 -11.345372 -1.070e+01 -9.4546843
## Pregnancies                0.092051   0.133581   0.157250  1.812e-01  0.2240009
## Glucose                    0.034571   0.040904   0.044371  4.789e-02  0.0550854
## Insulin                   -0.003896  -0.002406  -0.001673 -9.314e-04  0.0005335
## BMI                        0.089012   0.112268   0.125153  1.382e-01  0.1654697
## DiabetesPedigreeFunction   0.291858   0.800093   1.062822  1.336e+00  1.8650120
```

```
beta_means <- summary(glm.bayes2)$statistics[,1]
z = as.matrix(X_new) %*% as.matrix(beta_means)
y_pred <- 1/(1+exp(-z))

# Hit matrix and hit ratio (Ridge model 1-se)
H <- table(y_pred> 0.5, y_test == 1)
H
```

```
##
##         FALSE TRUE
##   FALSE   124   48
##   TRUE     18   41
```

```
tp = H[2,2]
tn = H[1,1]
fp = H[2,1]
fn = H[1,2]
total = tp+tn+fp+fn

acc = (tp+tn)/total
rec = (tp)/(tp+fp)
pre = (tp)/(tp+fn)
F1 = 2*rec*pre/(pre+rec)

print(paste('Accuracy: ',acc ))
```

```
## [1] "Accuracy:  0.714285714285714"
```

```
print(paste('Recall: ', rec))
```

```
## [1] "Recall:  0.694915254237288"
```

```
print(paste('Precision: ',pre))
```

```
## [1] "Precision:  0.460674157303371"
```

```
print(paste('F1-score: ',F1 ))
```

```
## [1] "F1-score:  0.554054054054054"
```

```
D = freq_glm$deviance
D0 = freq_glm$null.deviance

print(paste('R2: ', 1-D/D0 ))
```

```
## [1] "R2:  0.327058067272475"
```

The results remain very similar between the different Bayesian models and the frequentist models, this might be happening as the data is to strong, and is vanishing the weak priors that are fed to the model.

Conclusions

After performing several models for this data set, we have concluded:

1- For defining a useful Bayesian model it is key to have realistic and confident prior estimations about the model parameters. If that is not the case, the model ends up with very similar predictions as the frequentist model.

2- One of the benefits of using Bayesian models, is the flexibility of having a density function for every parameter, which allows many different predictions and even non-fixed predictions of the target variable.

3- For generalized linear models, Bayesian model selection is not well deployed yet. The best option in this case, it to perform model selection on the frequentist paradigm (by doing Lasso, stepwise, etc.) and then compute the Bayesian model with those selected variables.

4- Models for this data set are performing poorly, with moderate accuracy (majority class is about 66 %) and very low precision. One cause for this problem, might be the unbalanced problem of the target variable. For optimizing this problem, *F1 score* could be tuned setting the right threshold.