Resampling Methods and Bayesian Models

Charlie Qu

NOV 12 2017

Part I.
Construct the data

```r
# set the random seed so that we can replicate results.
set.seed(8675309)
```

```r
# true parameters
sigma = 2.5
betatrue = c(4,2,0,0,0,-1,0,1.5, 0,0,0,1,0,.5,0,0,0,0,-1,1,4)
#          int|    X1                            | X2     |X3

truemodel = betatrue != 0
```

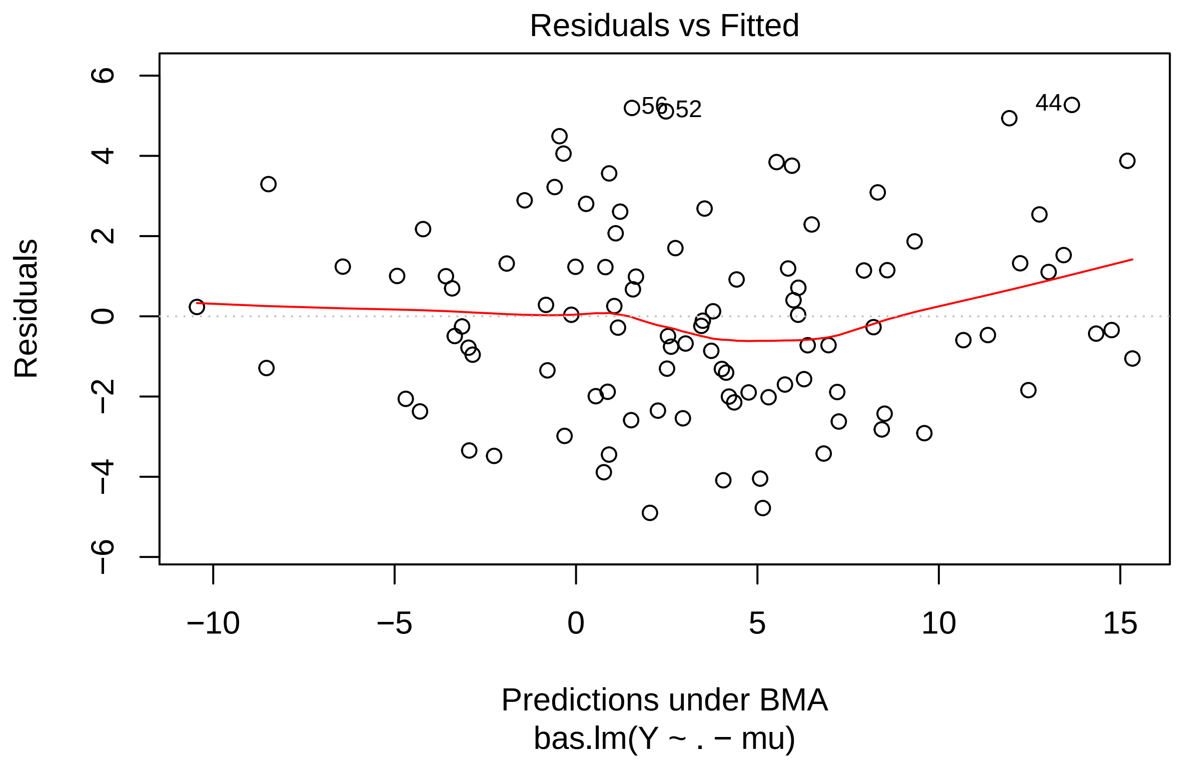
```r
#sample size
n = 1000

# generate some standard normals
  Z = matrix(rnorm(n*10, 0, 1), ncol=10, nrow=n)

#  Create X1 by taking linear cominations of Z to induce correlation among X1 components

  X1 = cbind(Z,
            (Z[,1:5] %*% c(.3, .5, .7, .9, 1.1) %*% t(rep(1,5)) +
            matrix(rnorm(n*5, 0, 1), ncol=5, nrow=n))
            )
# generate X2 as a standard normal
  X2 <- matrix(rnorm(n*4,0,1), ncol=4, nrow=n)

# Generate X3 as a linear combination of X2 and noise
  X3 <- X2[,4]+rnorm(n,0,sd=0.1)

# combine them
  X <- cbind(X1,X2,X3)

# Generate mu
# X does not have a column of ones for the intercept so need to add the intercept
# for true mu
mu = betatrue[1] + X %*% betatrue[-1]

# now generate Y
Y = mu + rnorm(n,0,sigma)

# make a dataframe and save it
df = data.frame(Y, X, mu)
```

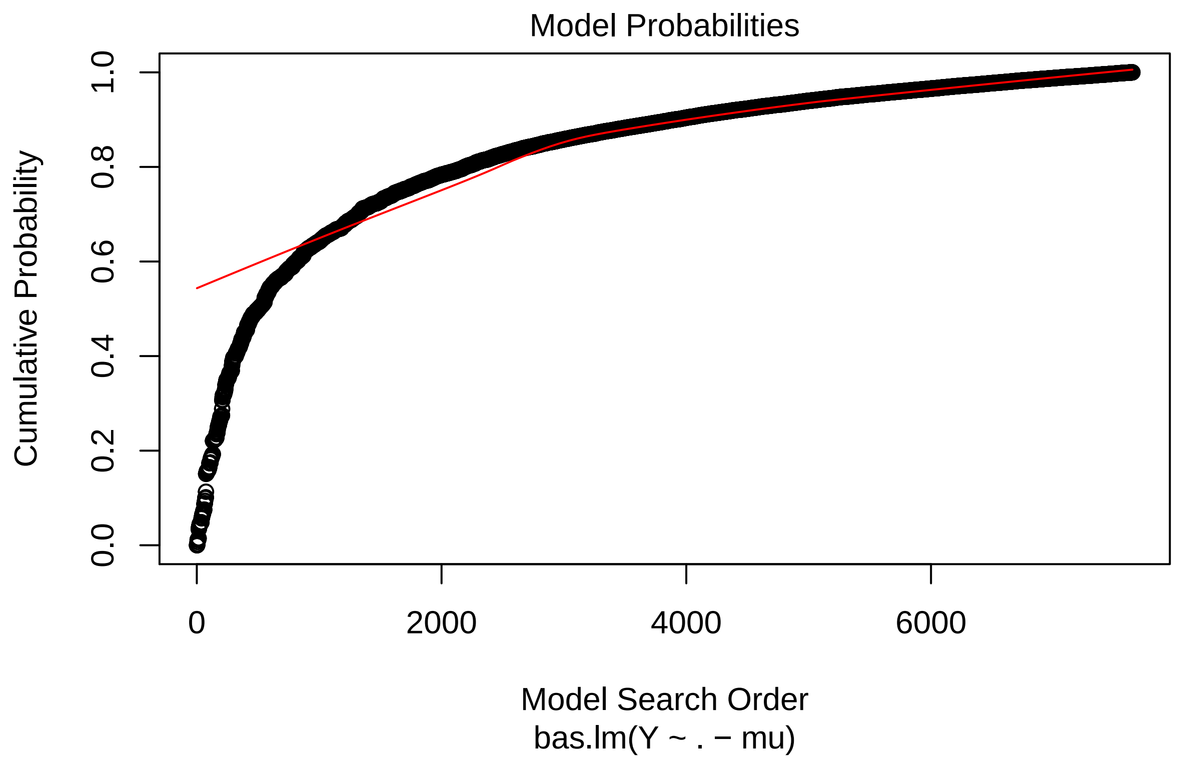
Split the data set into a training set containing 100 observations and a test set containing 900 observations. Set the random seed based on team number:

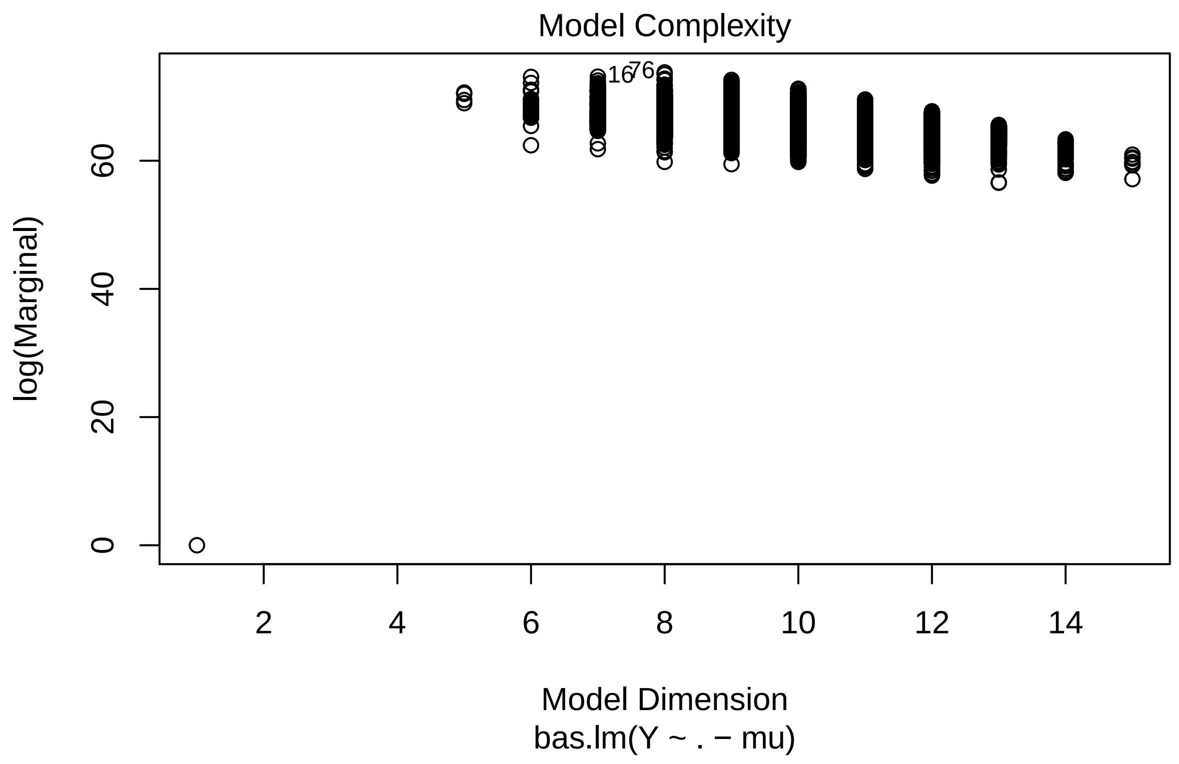
```r
set.seed(11)
n = nrow(df)
n.train = floor(.10*n)
train = sample(1:n, size=n.train, replace=FALSE)
df.train = df[train,]
df.test = df[-train,]
```

1. Obtain the posterior distribution using the g-prior with $g = n$ for the training data. If you do not enumerate and use MCMC, please include diagnostic plots to indicate that the MCMC has run long enough. *Hint: use `cache=T` in the chunk to save the output so that you do not need to enumerate of rerun the MCMC everytime you update other chunks of code.*

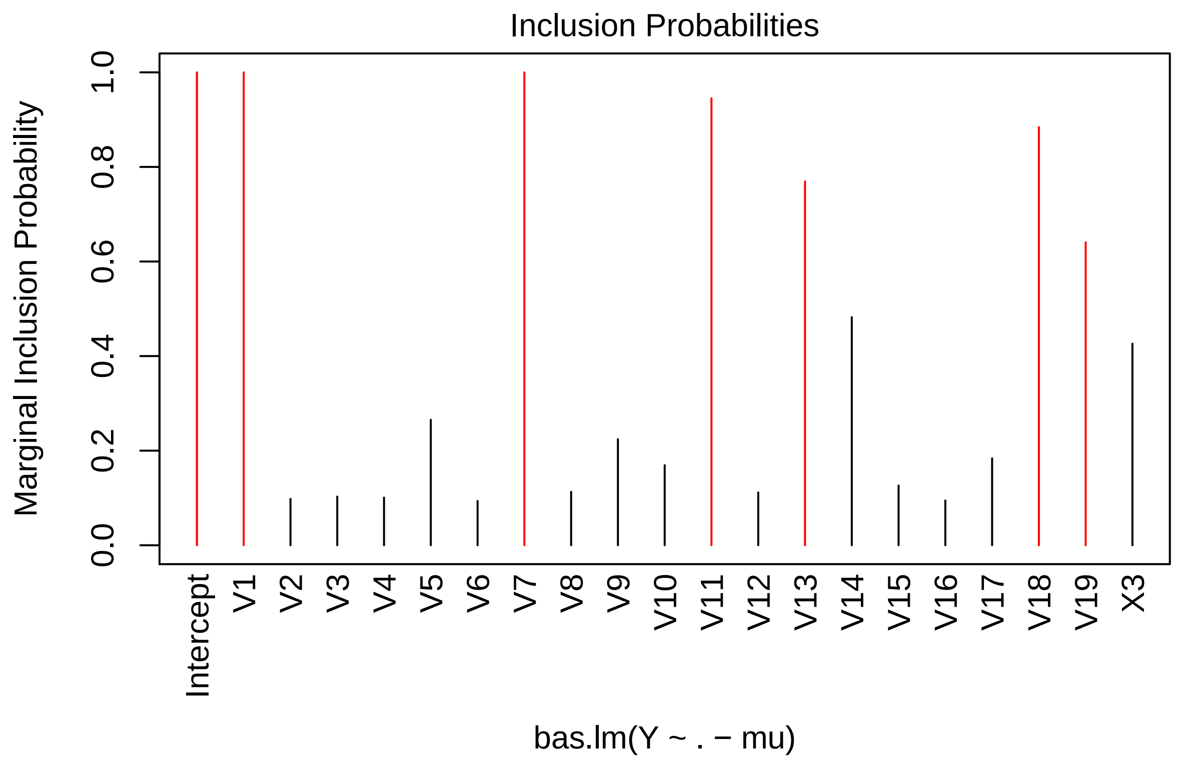
```r
df.bas =  bas.lm(Y ~ . - mu, data=df.train,
                 prior="g-prior", a=nrow(df.train), modelprior=uniform(),
                 method="MCMC", MCMC.iterations = 3000000, thin = 20)
plot(df.bas)
```




Model Probabilities
Cumulative Probability
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
Model Search Order
bas.lm(Y ~ . − mu)

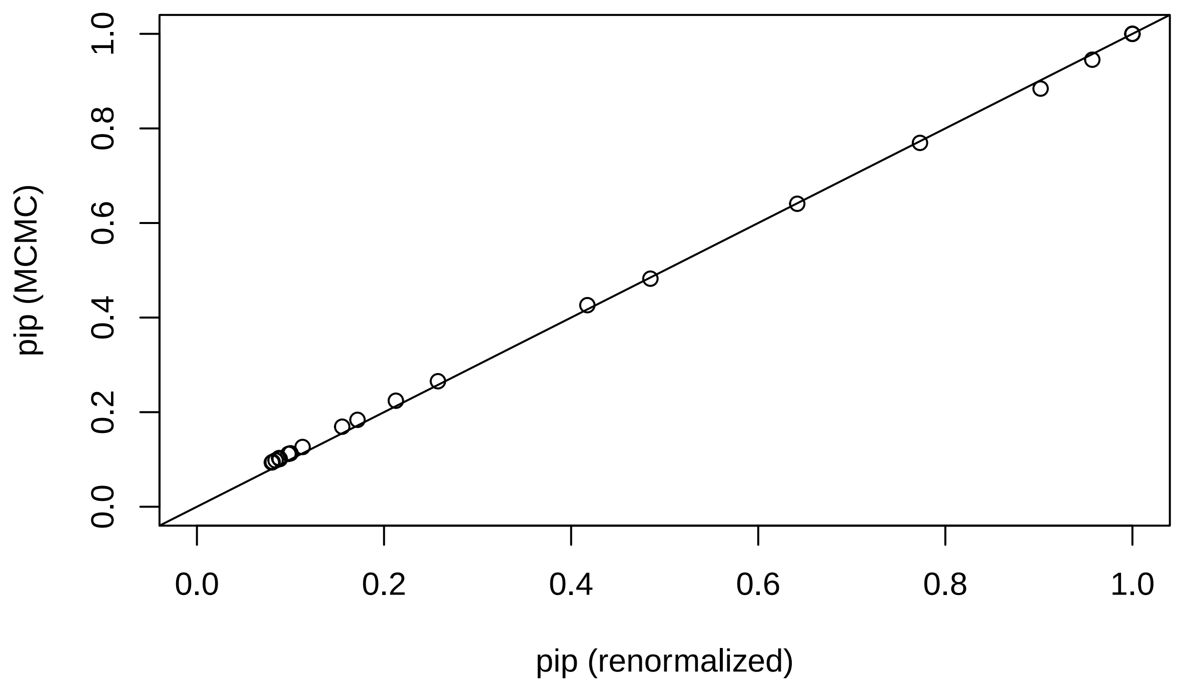


Model Complexity
log(Marginal)
0
20
40
60
2
4
6
8
10
12
14
16
76
Model Dimension
bas.lm(Y ~ . − mu)



Diagnosis:

```
diagnostics(df.bas, type="pip")
```



From the model propability plot, we can see that the decresing of slope, indicating that a decrese of extra information. Therefore, our number of iteration is enought. From the last diagnosis plot, we can see that the pip(posterior inclusion probability) from MCMC is very close to the renormalized pip, also indicating we have enough number of iterations.

2. Compute the average RMSE for a) estimating β_{true}, b) estimating $\mu_{true} = X_{\text{test}}\beta_{true}$ and c) out of sample prediction of Y_{test} using BMA, the highest posterior probability model, (HPM) and the median probability model (MPM). Add the RMSE's to the table from HW5 and comment.

RMSE fucntion:

```r
#rmse function:
rmse=function(y,y.pred){
  rmse.val=sqrt(mean((y-y.pred)^2))
  return(rmse.val)
}
```

BMA

```r
#beta
betas.bas.BMA=coef(df.bas,estimator="BMA")
df.beta.BMA=data.frame(betatrue,betahat=betas.bas.BMA$postmean)
rmse.beta.BMA=rmse(df.beta.BMA$betatrue,df.beta.BMA$betahat)
rmse.beta.BMA
```

```
## [1] 0.6973722
```

```r
#mu
muhat.BMA = predict(df.bas, df.test, estimator="BMA",se.fit = T,prediction = F)
rmse.mu.BMA=rmse(df.test$mu,muhat.BMA$fit)
rmse.mu.BMA
```

```
## [1] 1.057787
```

```
#prediction
pred.BMA = predict(df.bas, df.test, estimator="BMA",se.fit = T,prediction = T)
rmse.pred.BMA=rmse(df.test$Y,pred.BMA$fit)
rmse.pred.BMA
```

```
## [1] 2.761119
```

HPM

```
#beta
betas.bas.HPM=coef(df.bas,estimator="HPM")
df.beta.HPM=data.frame(betatrue,betahat=betas.bas.HPM$postmean)
rmse.beta.HPM=rmse(df.beta.HPM$betatrue,df.beta.HPM$betahat)
rmse.beta.HPM
```

```
## [1] 1.250881
```

```
#mu
muhat.HPM = predict(df.bas, df.test, estimator="HPM",se.fit = T,prediction = F)
rmse.mu.HPM=rmse(df.test$mu,muhat.HPM$fit)
rmse.mu.HPM
```

```
## [1] 1.254176
```

```
#prediction
pred.HPM = predict(df.bas, df.test, estimator="HPM",se.fit = T,prediction = T)
rmse.pred.HPM=rmse(df.test$Y,pred.HPM$fit)
rmse.pred.HPM
```

```
## [1] 2.841517
```

MPM

```
#beta
betas.bas.MPM=coef(df.bas,estimator="MPM")
df.beta.MPM=data.frame(betatrue,betahat=betas.bas.MPM$postmean)
rmse.beta.MPM=rmse(df.beta.MPM$betatrue,df.beta.MPM$betahat)
rmse.beta.MPM
```

```
## [1] 1.241693
```

```
#mu
muhat.MPM = predict(df.bas, df.test, estimator="MPM",se.fit = T,prediction = F)
rmse.mu.MPM=rmse(df.test$mu,muhat.MPM$fit)
rmse.mu.MPM
```

```
## [1] 1.153132
```

```
#prediction
pred.MPM = predict(df.bas, df.test, estimator="MPM",se.fit = T,prediction = T)
rmse.pred.MPM=rmse(df.test$Y,pred.MPM$fit)
rmse.pred.MPM
```

```
## [1] 2.770051
```

From HW5, we have:
for full model, RMSE of $\beta = 1.41319$, RMSE of $\mu = 1.05590$, RMSE of $Y = 2.72092$

for AIC model, RMSE of $\beta = 1.27390$, RMSE of $\mu = 0.90406$, RMSE of $Y = 2.66848$
for BIC model, RMSE of $\beta = 1.27400$, RMSE of $\mu = 1.015786$, RMSE of $Y = 2.72873$

```
table.rmse=rbind(c(1.41319,1.05590,2.72092),c(1.27390, 0.90406,2.66848),c(1.27400, 1.015786, 2.72873),c

colnames(table.rmse)=c("rmse.beta","rmse.mu","rmse.prediction")
rownames(table.rmse)=c("full.model","AIC","BIC","BMA","HPM","MPM")
kable(table.rmse)
```

	rmse.beta	rmse.mu	rmse.prediction
full.model	1.4131900	1.055900	2.720920
AIC	1.2739000	0.904060	2.668480
BIC	1.2740000	1.015786	2.728730
BMA	0.6973722	1.057787	2.761119
HPM	1.2508812	1.254176	2.841518
MPM	1.2416927	1.153132	2.770051

Discussion:
From the table, we notice that, from the perspective of prediction, bayes regression performs worse than what we did in HW5:AIC,BIC and full model. However, in terms of finding the true model, bayes regression are generally better than AIC and BIC. Inparticular, BMA did a excellent job in terms of finding the true model with a rmse of β equal to only 0.6973.

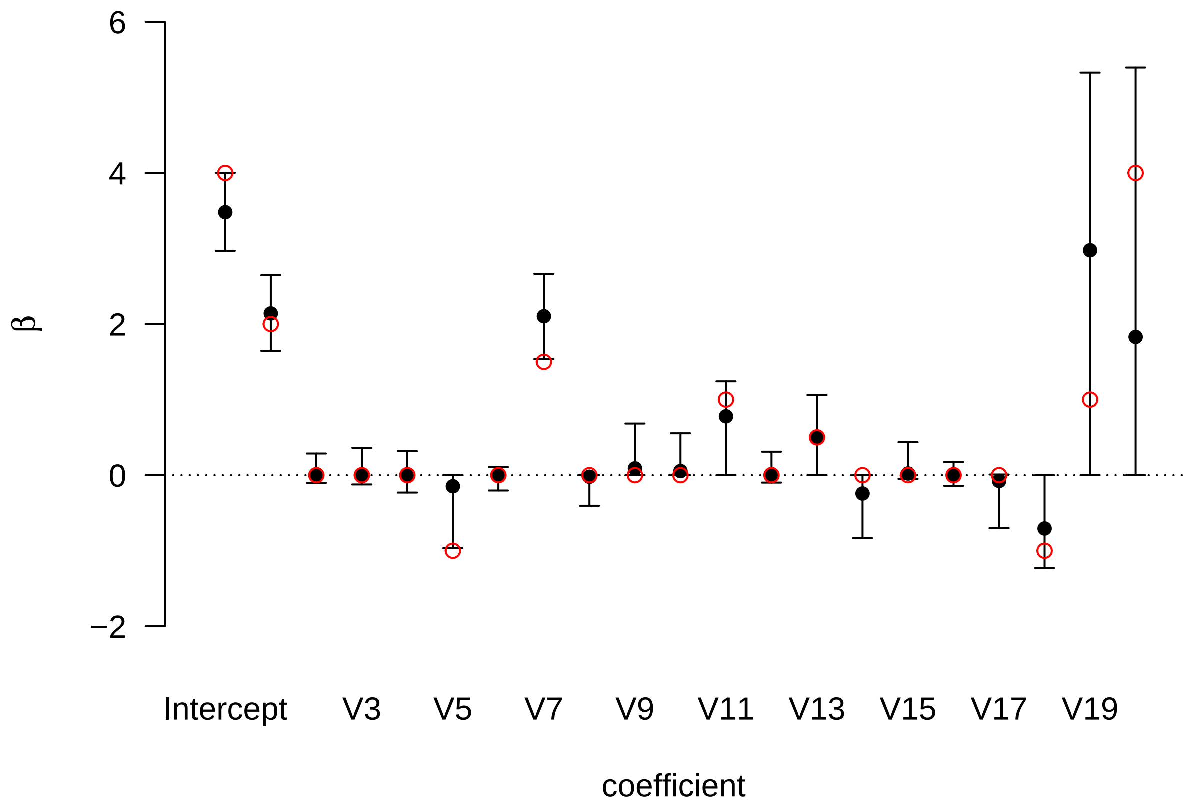
3. Provide plots that show confidence/prediction intervals for β, and μ, and Y in the test data under BMA, HPM, and MPM, with the true data added. Provide a summary table that summarizes what percent of the intervals contain the true values and add to your results from HW5.

BMA

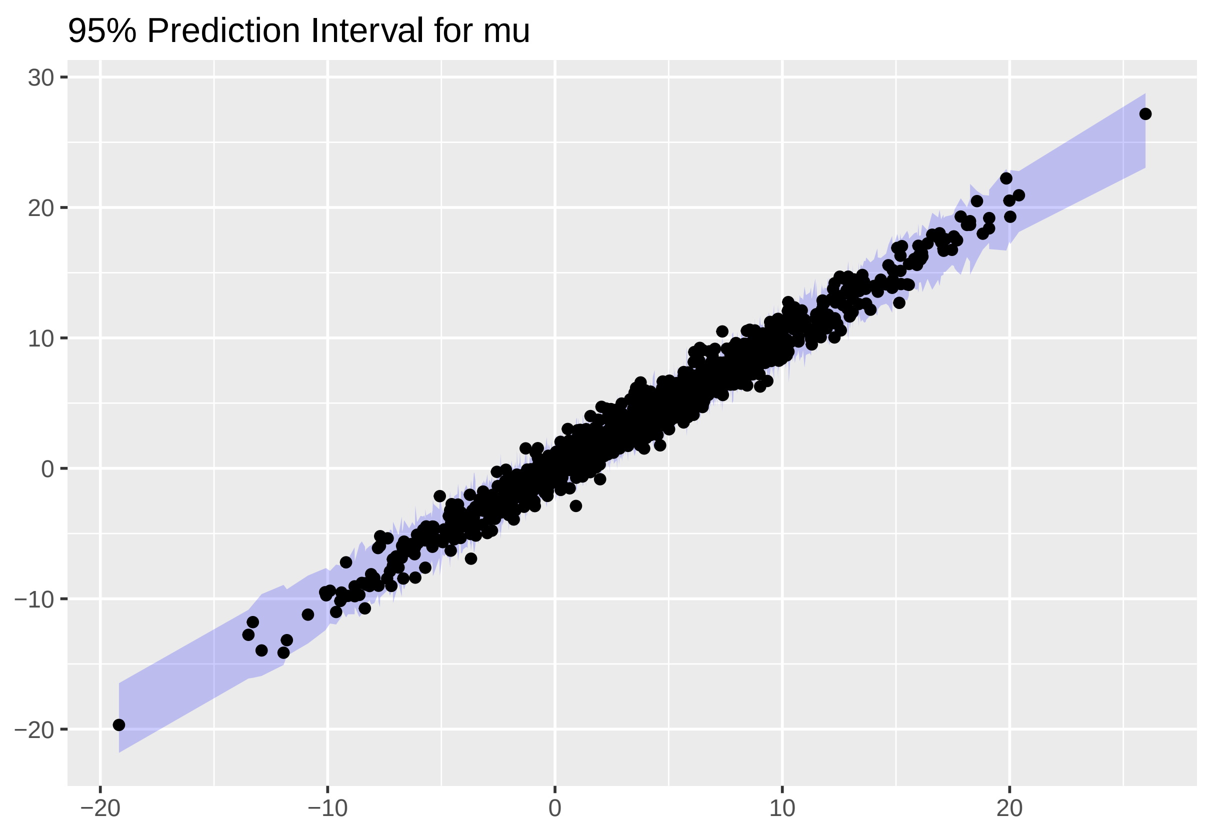
```
#cover function
cover=function(x){
  return(mean(x[,3]>x[,1] & x[,3]<x[,2]))
}



#confidence interval,beta
CI.beta.BMA=confint(betas.bas.BMA)
df.CI.beta.BMA=data.frame(L=CI.beta.BMA[,1],
                       U=CI.beta.BMA[,2],
                       V=betatrue)
plot(confint(betas.bas.BMA))
```

```
## NULL
```

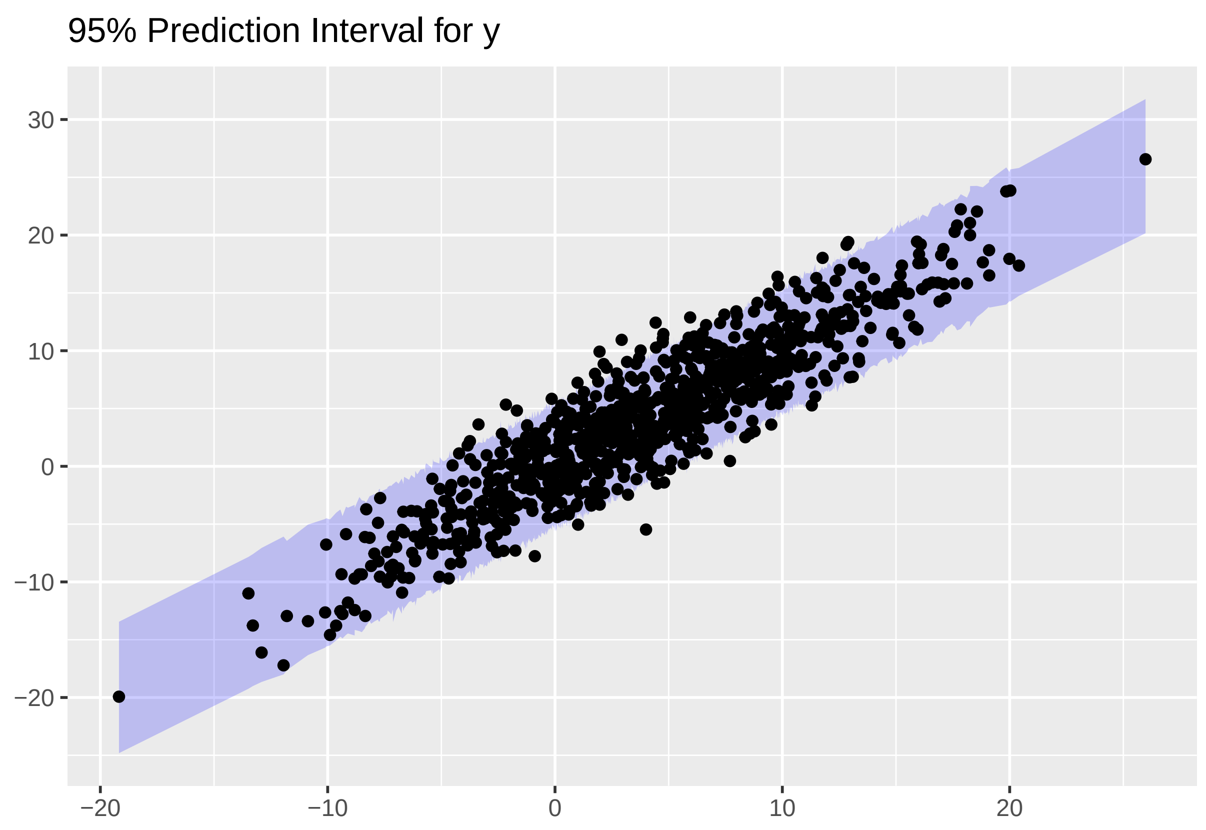
```
points(1:length(betatrue),betatrue,col=2)
```



```
#calculate percentage
perc.beta.BMA=cover(df.CI.beta.BMA)


#prediction interval,mu
CI.mu.BMA=confint(muhat.BMA,parm = "mean")
df.CI.mu.BMA=data.frame(O=df.test$mu,
                        P=CI.mu.BMA[,3],
                        L=CI.mu.BMA[,1],
                        U=CI.mu.BMA[,2])
ggplot(df.CI.mu.BMA, aes(x=P, y=O)) +
    geom_ribbon(aes(ymin = L, ymax = U), fill = "blue", alpha = 0.2) +
    geom_point(aes(y=O)) + xlab("") +
    ylab("") +
    ggtitle("95% Prediction Interval for mu")
```



```
#calculate percentage
perc.mu.BMA=cover(df.CI.mu.BMA[,c(3,4,1)])

#prediction interval,y
CI.y.BMA=confint(pred.BMA,parm = "pred")
df.CI.y.BMA=data.frame(O=df.test$Y,
                       P=CI.y.BMA[,3],
                       L=CI.y.BMA[,1],
                       U=CI.y.BMA[,2])
ggplot(df.CI.y.BMA, aes(x=P, y=O)) +
    geom_ribbon(aes(ymin = L, ymax = U), fill = "blue", alpha = 0.2) +
    geom_point(aes(y=O)) + xlab("") +
    ylab("") +
    ggtitle("95% Prediction Interval for y")
```



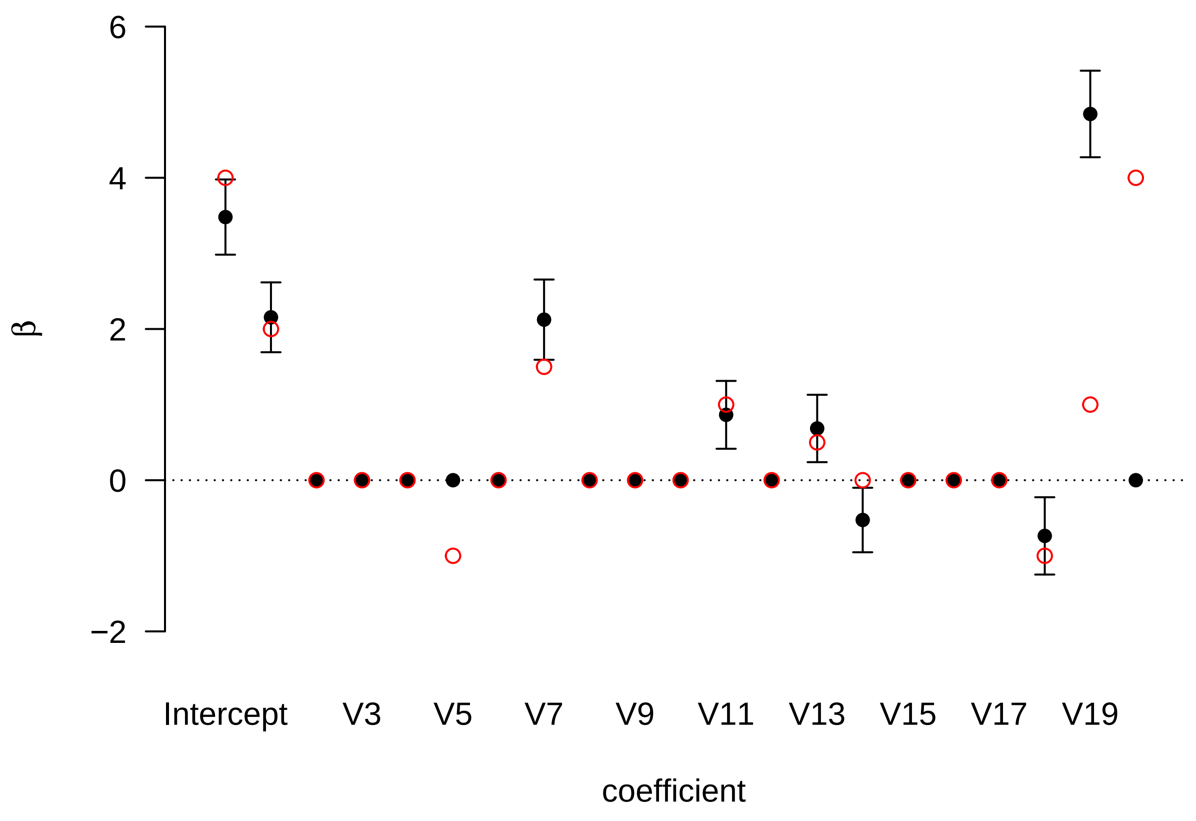
```r
#calculate percentage
perc.y.BMA=cover(df.CI.y.BMA[,c(3,4,1)])
```

HPM

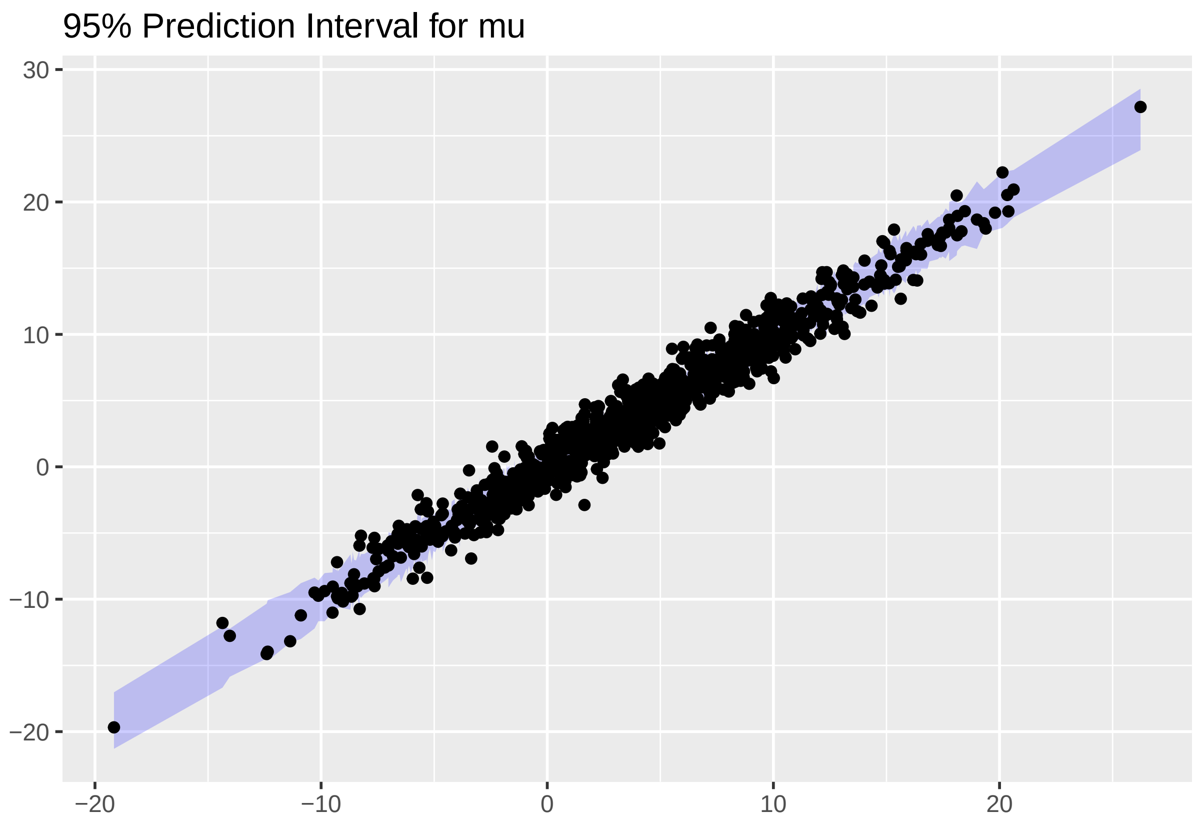
```r
#confidence interval,beta
CI.beta.HPM=confint(betas.bas.HPM)
df.CI.beta.HPM=data.frame(L=CI.beta.HPM[,1],
                          U=CI.beta.HPM[,2],
                          V=betatrue)
plot(confint(betas.bas.HPM))
```

```
## NULL
```

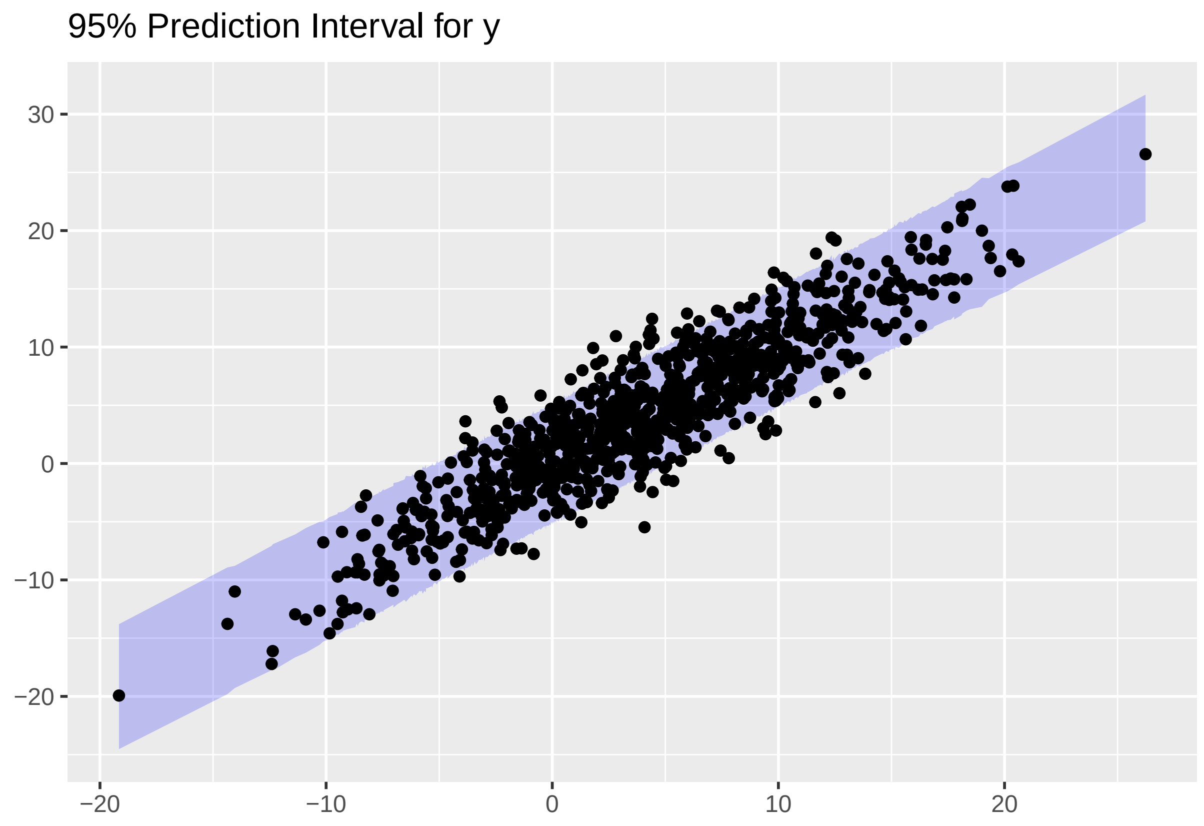
```r
points(1:length(betatrue),betatrue,col=2)
```



```
#calculate percentage
perc.beta.HPM=cover(df.CI.beta.HPM)


#prediction interval,mu
CI.mu.HPM=confint(muhat.HPM,parm = "mean")
df.CI.mu.HPM=data.frame(O=df.test$mu,
                       P=CI.mu.HPM[,3],
                       L=CI.mu.HPM[,1],
                       U=CI.mu.HPM[,2])
ggplot(df.CI.mu.HPM, aes(x=P, y=O)) +
    geom_ribbon(aes(ymin = L, ymax = U), fill = "blue", alpha = 0.2) +
    geom_point(aes(y=O)) + xlab("") +
    ylab("") +
    ggtitle("95% Prediction Interval for mu")
```



```
#calculate percentage
perc.mu.HPM=cover(df.CI.mu.HPM[,c(3,4,1)])

#prediction interval,y
CI.y.HPM=confint(pred.HPM,parm = "pred")
df.CI.y.HPM=data.frame(O=df.test$Y,
                       P=CI.y.HPM[,3],
                       L=CI.y.HPM[,1],
                       U=CI.y.HPM[,2])
ggplot(df.CI.y.HPM, aes(x=P, y=O)) +
    geom_ribbon(aes(ymin = L, ymax = U), fill = "blue", alpha = 0.2) +
    geom_point(aes(y=O)) + xlab("") +
    ylab("") +
    ggtitle("95% Prediction Interval for y")
```



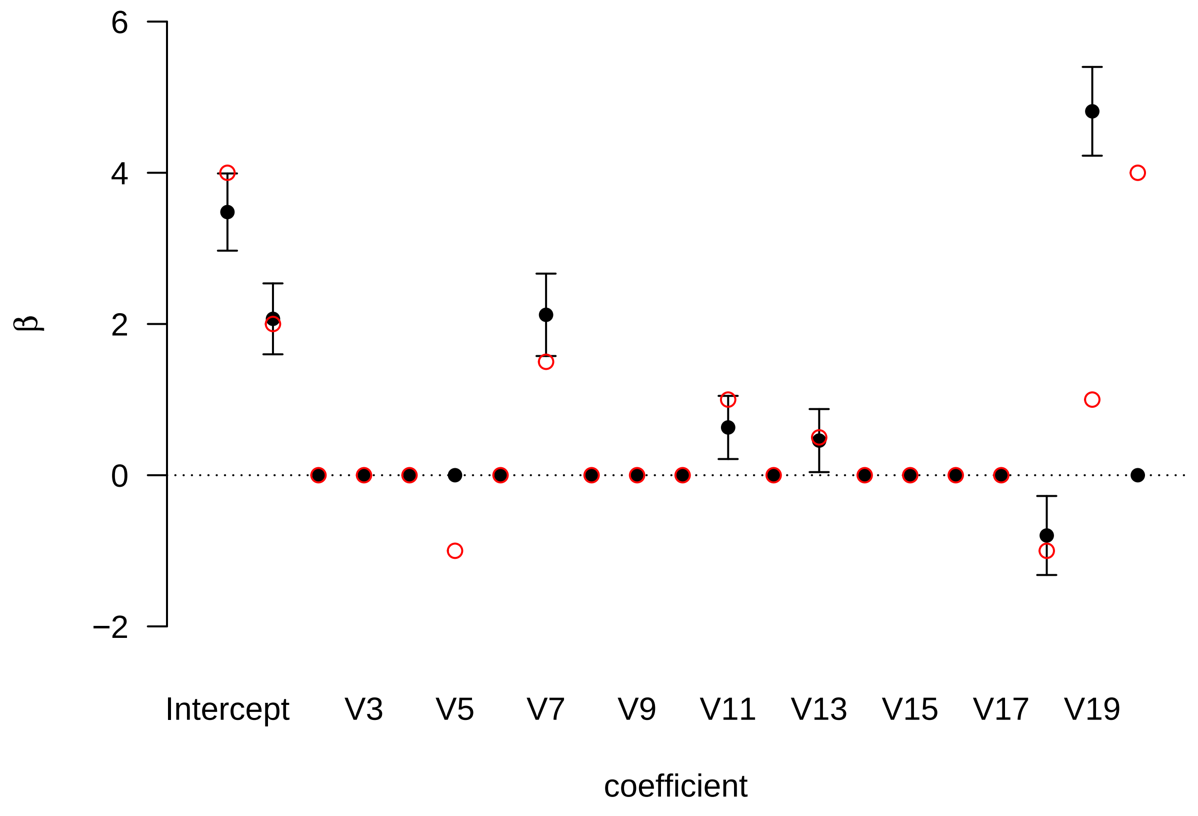
```
#calculate percentage
perc.y.HPM=cover(df.CI.y.HPM[,c(3,4,1)])
```

MPM

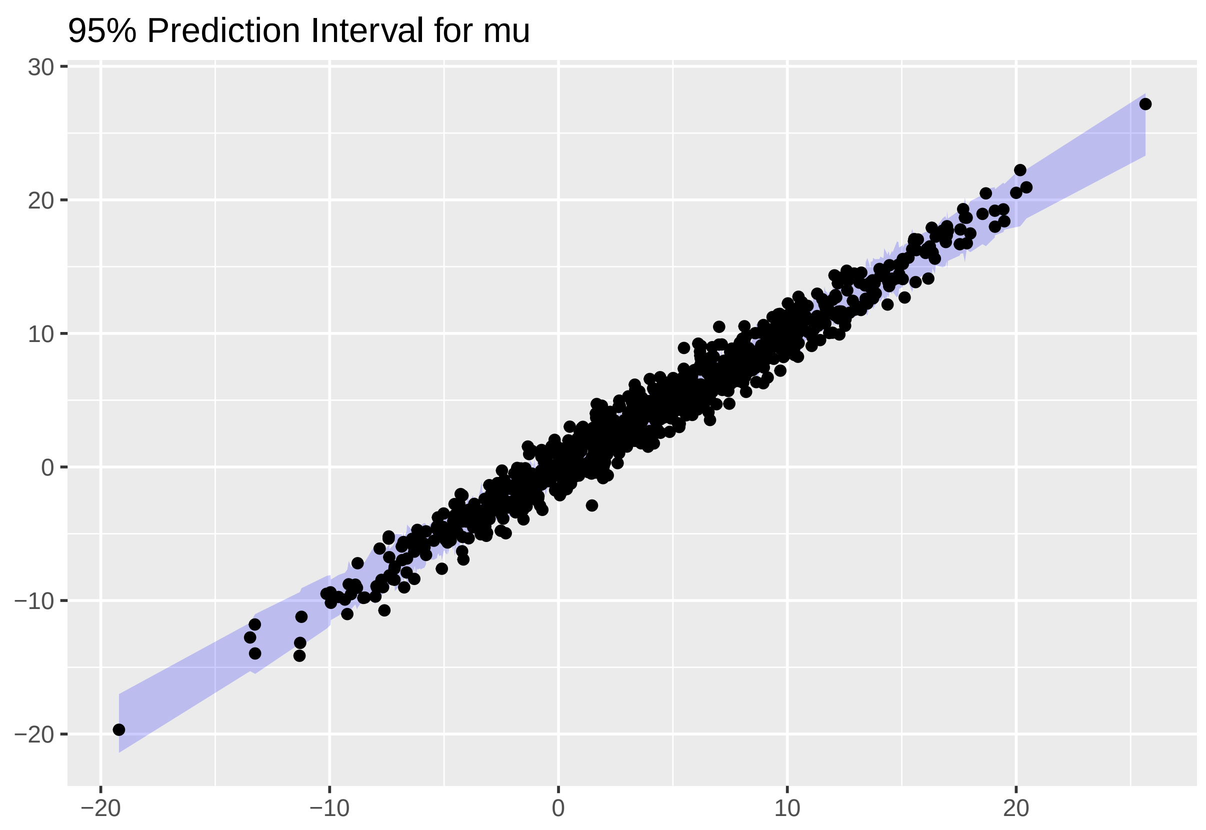
```
#confidence interval,beta
CI.beta.MPM=confint(betas.bas.MPM)
df.CI.beta.MPM=data.frame(L=CI.beta.MPM[,1],
                          U=CI.beta.MPM[,2],
                          V=betatrue)
plot(confint(betas.bas.MPM))
```

```
## NULL
```

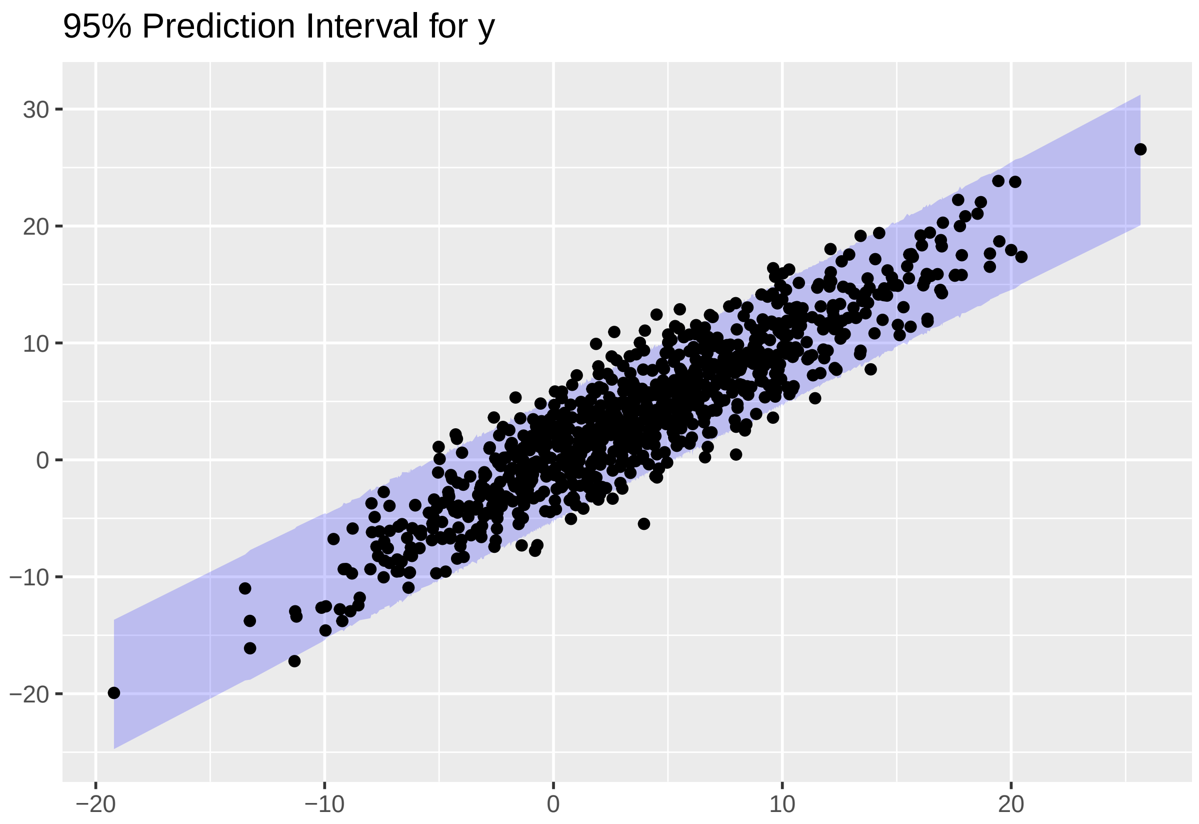
```
points(1:length(betatrue),betatrue,col=2)
```



```
#calculate percentage
perc.beta.MPM=cover(df.CI.beta.MPM)


#prediction interval,mu
CI.mu.MPM=confint(muhat.MPM,parm = "mean")
df.CI.mu.MPM=data.frame(O=df.test$mu,
                        P=CI.mu.MPM[,3],
                        L=CI.mu.MPM[,1],
                        U=CI.mu.MPM[,2])
ggplot(df.CI.mu.MPM, aes(x=P, y=O)) +
    geom_ribbon(aes(ymin = L, ymax = U), fill = "blue", alpha = 0.2) +
    geom_point(aes(y=O)) + xlab("") +
    ylab("") +
    ggtitle("95% Prediction Interval for mu")
```



```r
#calculate percentage
perc.mu.MPM=cover(df.CI.mu.MPM[,c(3,4,1)])

#prediction interval,y
CI.y.MPM=confint(pred.MPM,parm = "pred")
df.CI.y.MPM=data.frame(O=df.test$Y,
                       P=CI.y.MPM[,3],
                       L=CI.y.MPM[,1],
                       U=CI.y.MPM[,2])
ggplot(df.CI.y.MPM, aes(x=P, y=O)) +
    geom_ribbon(aes(ymin = L, ymax = U), fill = "blue", alpha = 0.2) +
    geom_point(aes(y=O)) + xlab("") +
    ylab("") +
    ggtitle("95% Prediction Interval for y")
```



```r
#calculate percentage
perc.y.MPM=cover(df.CI.y.MPM[,c(3,4,1)])
```

From HW5, we have:
for full model, coverage of $\beta = 100\%$, coverage of $\mu = 99.89\%$, Coverage of $Y = 98.22\%$
for AIC model, Coverage of $\beta = 89\%$, Coverage of $\mu = 85.22\%$, Coverage of $Y = 93.24\%$
for BIC model, Coverage of $\beta = 75\%$, Coverage of $\mu = 85.43\%$, Coverage of $Y = 93.89\%$

```r
table.cover=rbind(c(100,99.89,98.22)/100,c(89, 85.22,93.24)/100,c(75, 85.43, 93.89)/100,c(perc.beta.BMA

colnames(table.cover)=c("perc.beta","perc.mu","perc.prediction")
rownames(table.cover)=c("full.model","AIC","BIC","BMA","HPM","MPM")
kable(table.cover)
```

	perc.beta	perc.mu	perc.prediction
full.model	1.0000000	0.9989000	0.9822000
AIC	0.8900000	0.8522000	0.9324000
BIC	0.7500000	0.8543000	0.9389000
BMA	0.7619048	0.9600000	0.9511111
HPM	0.1904762	0.7233333	0.9288889
MPM	0.1904762	0.7444444	0.9366667

4. Summarize your findings comparing model averaging and selection with the methods from before and any recommendations.

Summarazation:
From the rmse table, we notice that, from the perspective of prediction, bayes regression performs worse than

what we did in HW5:AIC,BIC and full model. However, in terms of finding the true model, bayes regression are generally better than AIC and BIC. Inparticular, BMA did a excellent job in terms of finding the true model with a rmse of β equal to only 0.6973.

From the perctage table, we notice that the percentages of Bayesian Regression are generally lower than those of AIC, BIC and full model. And among Bayesian Regressions, the the percentages of HPM and MPM are much lower than those of BMA. In particular, the percentage for β of HPM and MPM are extremely. Therefore, this is consistant with the fact that MPM and Hpm are not suitable for data where the variables are highly correlated with each other.

5. Refer to the `Boston` housing data in the MASS library

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
data(Boston)
```

Using Bayesian regression, predict per capita crime (transforming any variables if needed based on any EDA). Provide a write up of your analysis describing the data, your model, which priors you used (and why), and interpretation of estimates for at least 5 important variables, paying careful attention to provide interval estimates or posterior distributions in addition to point testimates.

```
set.seed(1234)

RMSE=rmse
data(Boston)
require("car")
```

```
## Loading required package: car

##
## Attaching package: 'car'

## The following object is masked from 'package:dplyr':
##
##     recode
```

```
powerTransform(Boston + .01)
```

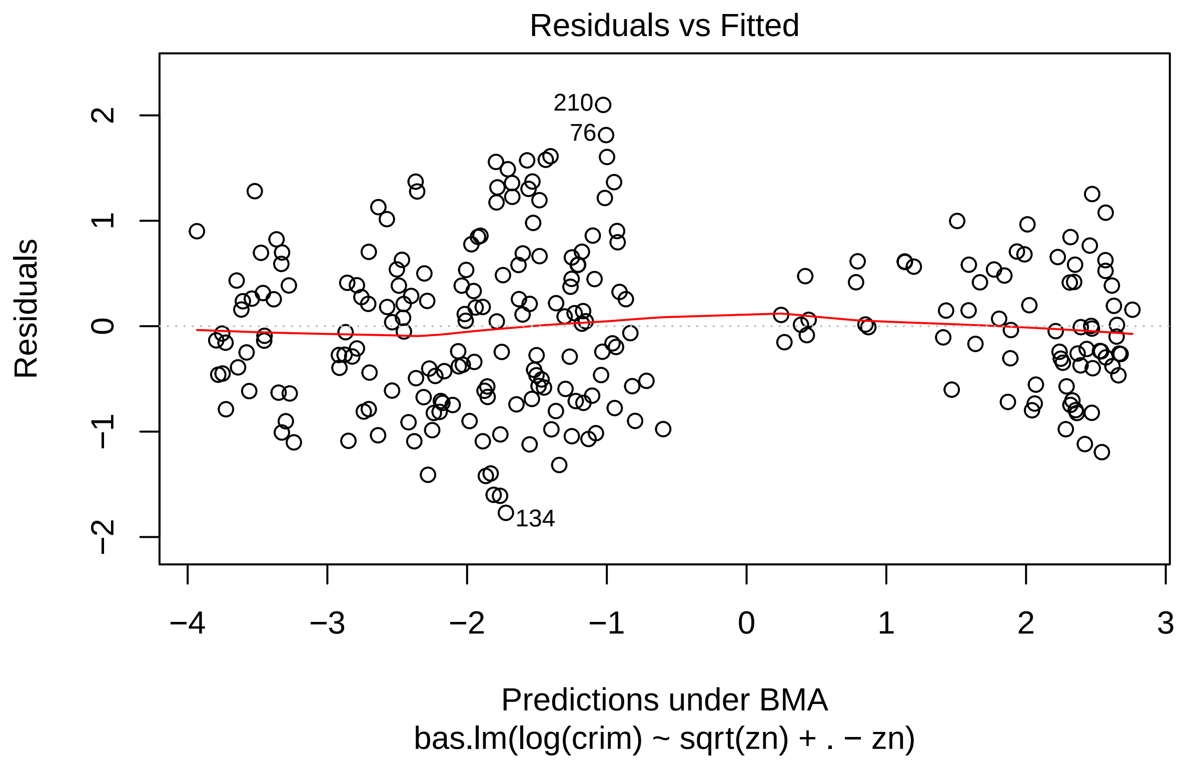
```
## Estimated transformation parameters
##        crim          zn       indus        chas         nox          rm
## -0.03576521 -0.39717763  0.51689118 -3.13293868 -1.48786856  1.26398472
##         age         dis         rad         tax     ptratio       black
##  1.52028925 -0.03342214  0.32391432  0.70820919  3.52181453  3.65426519
##       lstat        medv
##  0.21380647  0.40476479
```

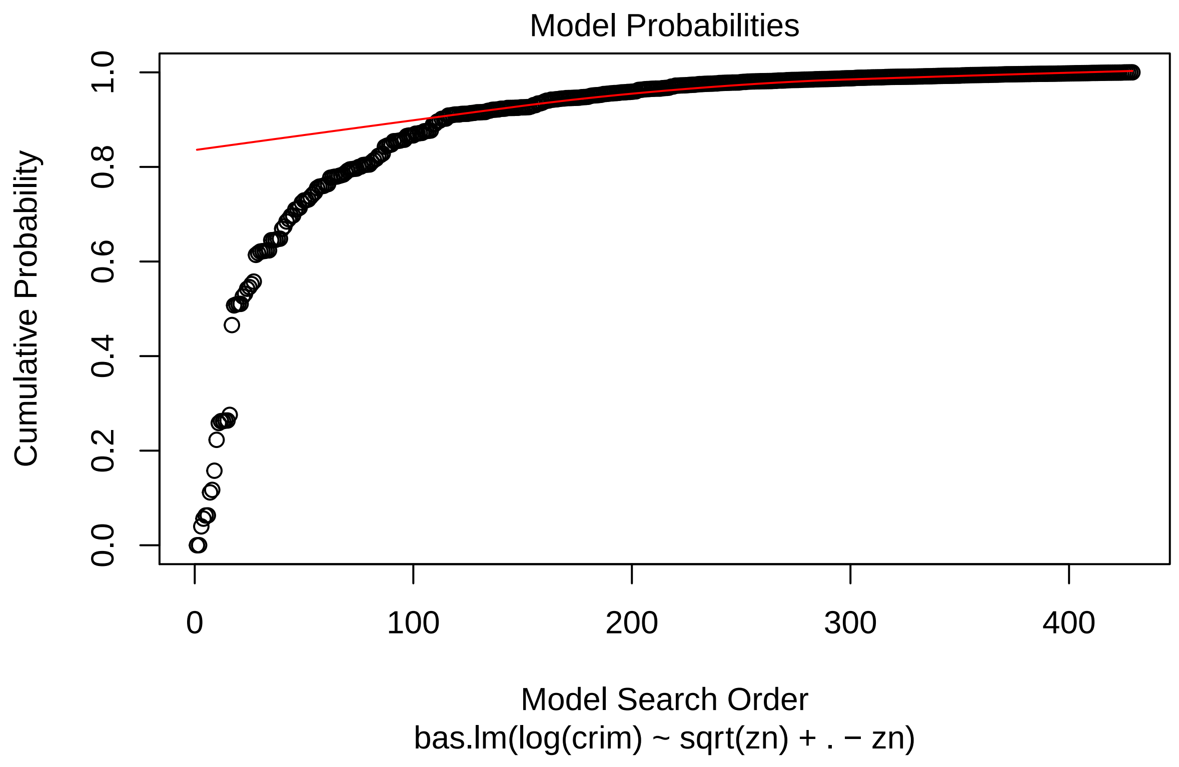
```
Boston = Boston %>% filter(crim < 50)

n = nrow(Boston)
n.train = 0.5 * n
train = sample(1:n, n.train, replace = FALSE)
Boston_train = Boston[train,]
Boston_test = Boston[-train,]
```

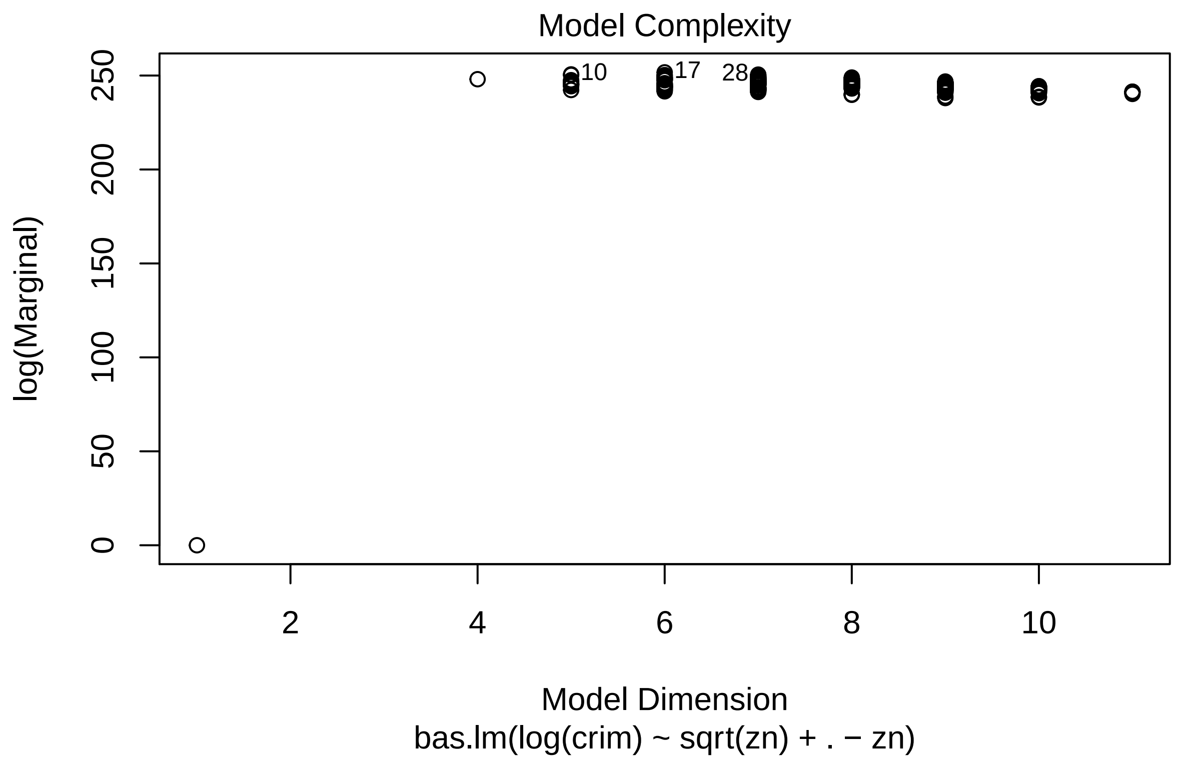
```r
crime_test <- Boston_test$crim

Boston_lm <- lm(crim~., data=Boston_train)
crime_test.Boston_lm <- predict(Boston_lm, newdata = Boston_test)

RMSE(crime_test, crime_test.Boston_lm)
```

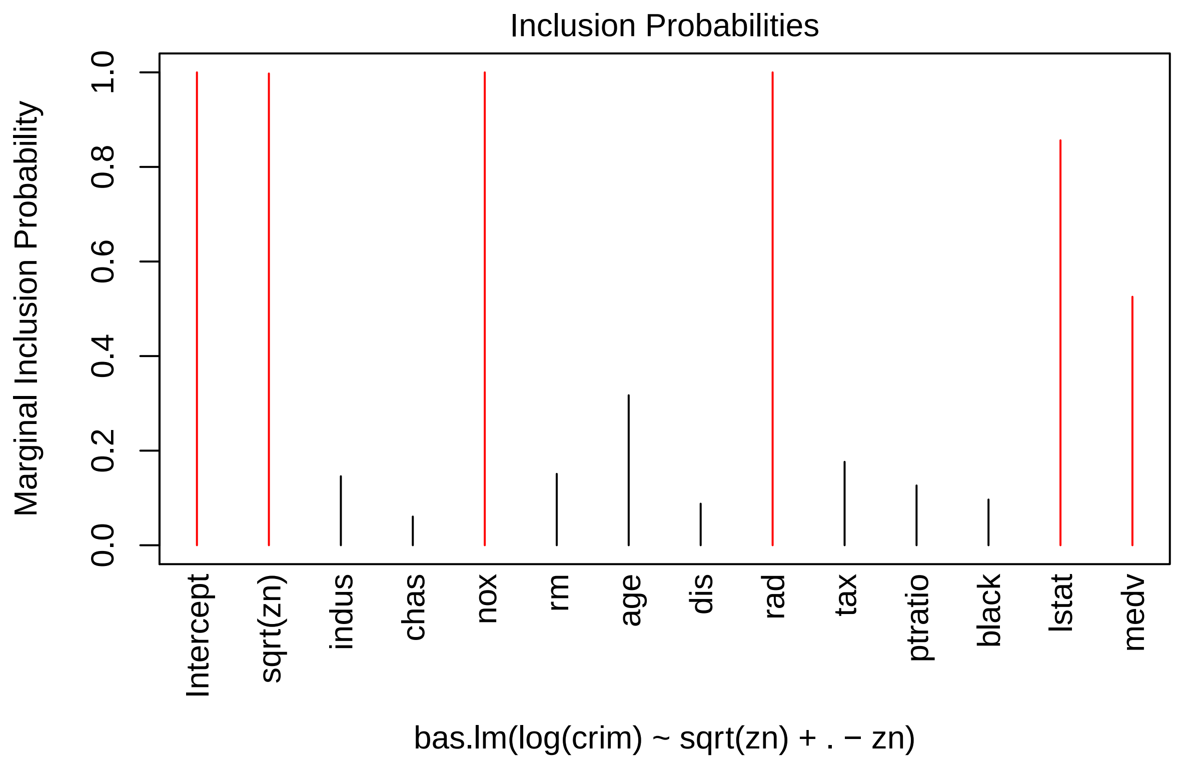
```
## [1] 4.002803
```

```r
Boston_model <- bas.lm(log(crim) ~ sqrt(zn) + . - zn, data = Boston_train,
                prior="g-prior", a=nrow(Boston_train), modelprior=uniform(),
                method="MCMC", MCMC.iterations = 1000000, thin = 20)

plot(Boston_model)
```

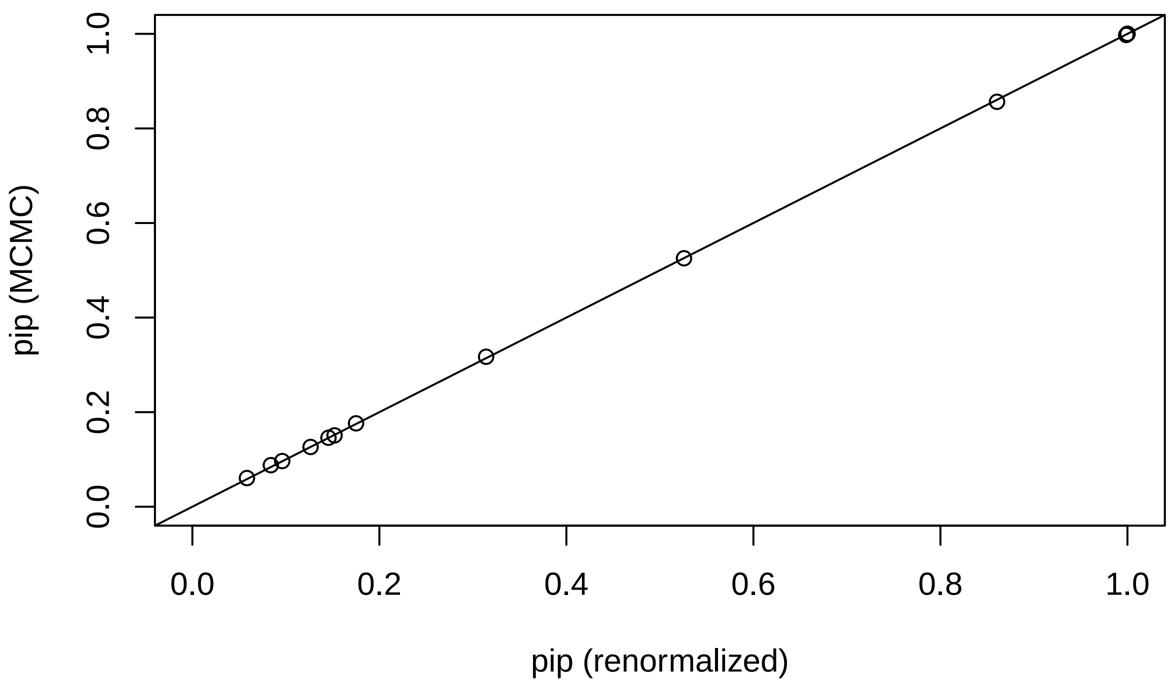


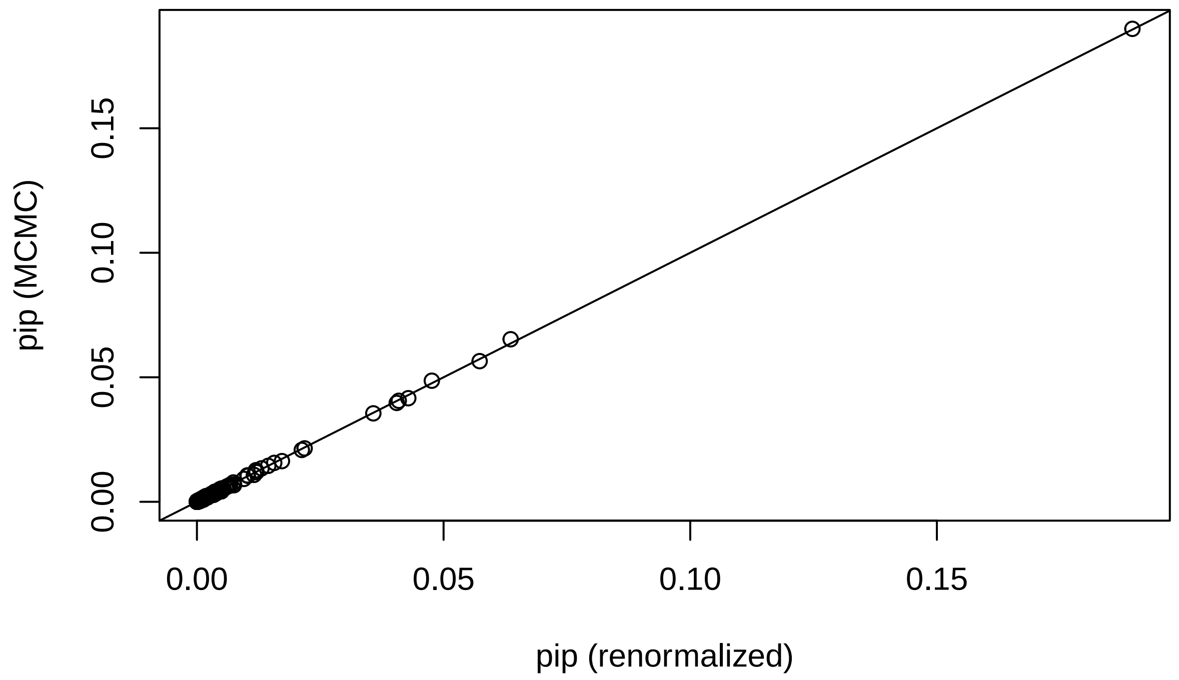

Model Probabilities
Cumulative Probability
0.0
0.2
0.4
0.6
0.8
1.0
0
100
200
300
400
Model Search Order
bas.lm(log(crim) ~ sqrt(zn) + . − zn)


Model Complexity
log(Marginal)
0
50
100
150
200
250
2
4
6
8
10
10
17
28
Model Dimension
bas.lm(log(crim) ~ sqrt(zn) + . − zn)

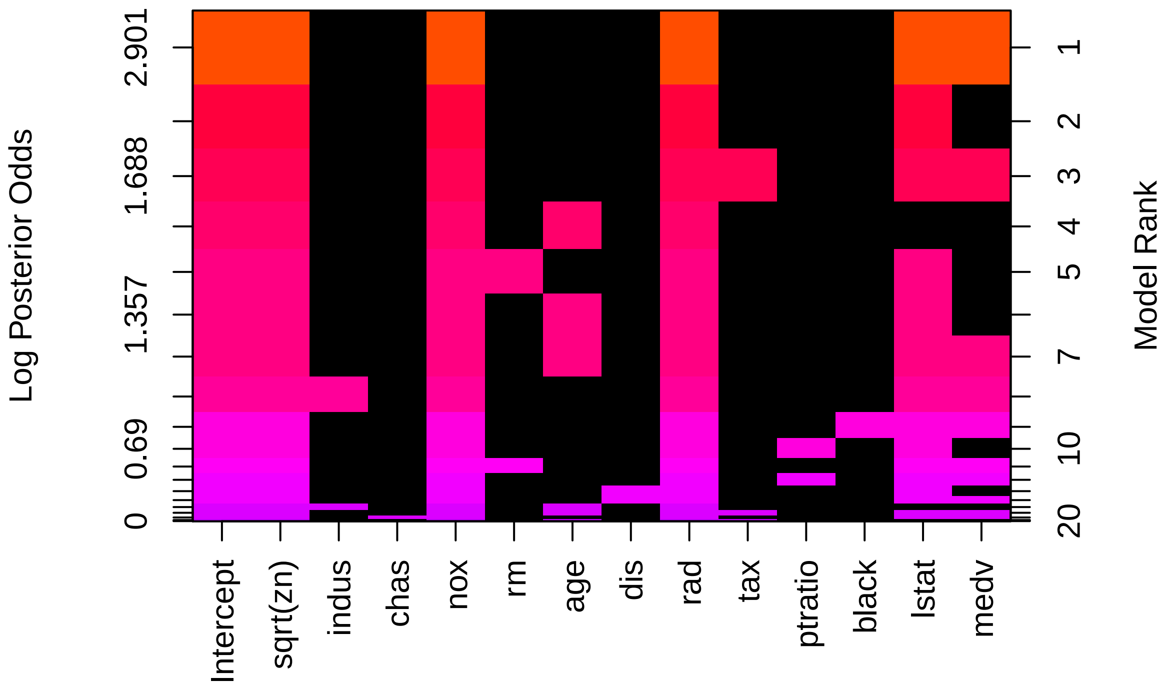


```r
diagnostics(Boston_model)
```


pip (MCMC)
0.0
0.2
0.4
0.6
0.8
1.0
pip (renormalized)



```r
image(Boston_model)
```



```
crime_test.Boston_model.BMA <- exp(predict(Boston_model, newdata = Boston_test, estimator = "BMA")$fit)
RMSE(crime_test, crime_test.Boston_model.BMA)
```

```
## [1] 4.04747
```

```
crime_test.Boston_model.HPM <- exp(predict(Boston_model, newdata = Boston_test, estimator = "HPM")$fit)
RMSE(crime_test, crime_test.Boston_model.HPM)
```

```
## [1] 4.011902
```

```
crime_test.Boston_model.MPM <- exp(predict(Boston_model, newdata = Boston_test, estimator = "MPM")$fit)
RMSE(crime_test, crime_test.Boston_model.MPM)
```

```
## [1] 4.011902
```