MATH2269 – Applied Bayesian Statistics

Final Project

- **Vishesh Jain**
  - S3666202
- **Abheer Sharma**
  - S3641618
- **Deepanshu Jain**
  - S3641651

Table of Contents

Introduction

In this report, Bayesian analysis has been used to predict a target variable using given predictors. Bayesian methodology is used to implement multiple linear regression to predict pm2.5 level in Beijing using 7 predictors. Bayesian methodology uses Monte-Carlo Chains to derive the values of coefficient of the linear regression equation.

Software/ Tools Used

To implement Bayesian analysis, the code has been developed in 'R Studios' which uses JAGS package to run Monte Carlo Markov Chains. The code also uses a source script provided by Dr. Haydar Demirhan as part of 'Applied Bayesian Statistics' course at RMIT University.

Data

This report presents the analysis of pm2.5 levels in Beijing using Bayesian methods. The dataset has been obtained from UCI Machine Learning Repository 'https://archive.ics.uci.edu/ml/datasets/Beijing+PM2.5+Data' . The given data has:

1 target variable

- Pm2.5: Particulate Matter 2.5 level in Beijing

Other variables:

- **No**: row number
- **year**: year of data in this row
- **month**: month of data in this row
- **day**: day of data in this row
- **hour**: hour of data in this row
- **pm2.5**: PM2.5 concentration (ug/m^3)
- **DEWP**: Dew Point (â„ƒ)
- **TEMP**: Temperature (â„ƒ)
- **PRES**: Pressure (hPa)
- **cbwd**: Combined wind direction
- **Iws**: Cumulated wind speed (m/s)
- **Is**: Cumulated hours of snow
- **Ir**: Cumulated hours of rain

Data Preparation

As part of the data processing or data preparation, following steps were taken:

- Cbwd – Cumulated wind speed
    - This variable is given as character or factor variable
    - It is given values as -2,-1,1,2 values for each level
- Variables No, year, month, day and hour are removed from the dataset as these are not to be used in linear regression. When time variables are used, it is treated as Time-Series Analysis and No act as ID of each row.
- The missing values exist in the dataset, but they are only 4.7% and hence are removed from the data set
- In the last step, the dataset is sorted so cbwd comes as the last column

Methods

To implement bayesian methods, normal distribution of 'pm2.5' has been assumed, where both mean (mu) and variance (sigma2) are unknown.

We will first proceed by building a JAGS model diagram presenting the chosen distributions of target variable (y) and its parameters (mu and sigma).

Then the analysis is done in using Markov chains on the distributions of the parameters. For adjusting the settings of JAGS model, we used random sample of size 500 from our data and when the diagnostics are good, we executed the chains with same settings on the whole data set.

NOTE: For simplifying the analysis and decreasing the use of high scale values and number of iterations, we have reduced the scale of target variable. We are using *pm2.5 in thousands*.

Implementation

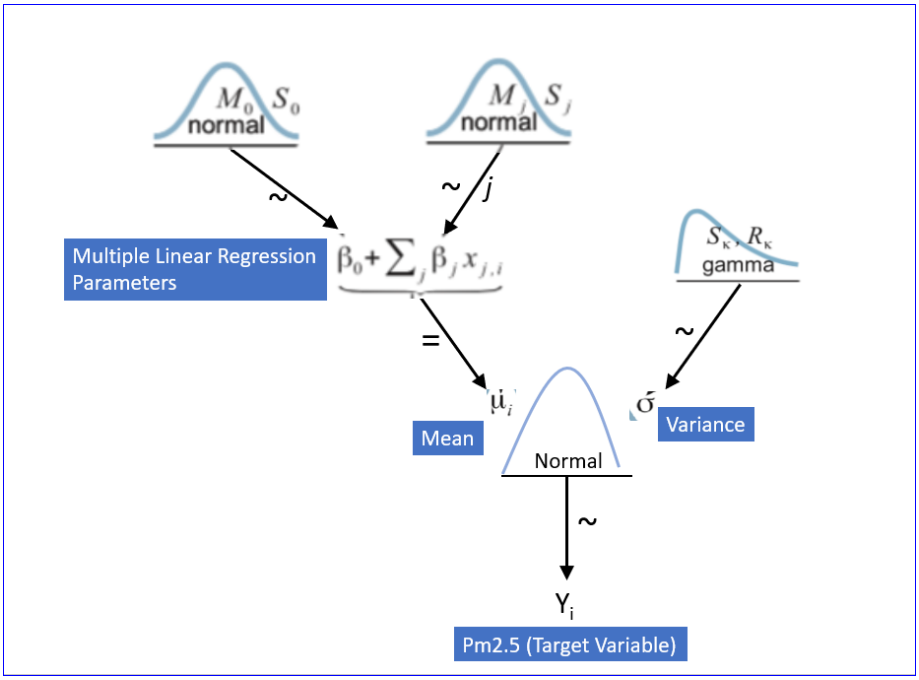
In this part, we will use Bayesian methods to predict pm2.5 values based on given values of the predictors. Here we no prior information about any predictors.

Multiple Linear Regression will be used to predict the pm2.5 for:

DEWP	TEMP	PRES	Iws	Is	Ir	cbwd
-11	-5	1025	1.57	1	0	-2
20	15	1000	2.5	0	0	1

Here, the cbwd values are taken as -2 and 1 for defining levels of factor variable

Model Diagram



In the model diagram we have defined mu (mean) as a multiple linear regression function. We have further defined the distribution of the linear regression parameters and not the variables as Normal (M, S).

Code for JAGS Model

In this part of the analysis, we are using run.jags() function from 'runjags' package.

We have also standardised all the predictor variables to have mean = 0 and variance = 1, with pm2.5 in thousands.

```
modelString = "
# Standardize the data:
data {
  ym <- mean(y)
  ysd <- sd(y)
  for ( i in 1:Ntotal ) {
    zy[i] <- ( y[i] - ym ) / ysd
  }
  for ( j in 1:((Nx-1)) ) {
    xm[j] <- mean(x[,j])
    xsd[j] <- sd(x[,j])
    for ( i in 1:Ntotal ) {
      zx[i,j] <- ( x[i,j] - xm[j] ) / xsd[j]
    }
  }

  # Specify the priors for original beta parameters
  # Prior locations to reflect the expert information
  # pm2.5 in thousands taken for the model to execute
  # Sample mean = 0.09861321
  mu0 <- ym # Set to overall mean a priori based on the interpretation of constant term in regression
  mu[1] <- 0.01 # DEWP - Dew Point
  mu[2] <- 0.01 # TEMP - Temperature
  mu[3] <- 0.01 # PRES - Pressure
  mu[4] <- 0.01 # lws - Cumulated Wind Speed
  mu[5] <- 0.01 # ls - Cumulated Hours of Snow
  mu[6] <- 0.01 # lr - Cumulated Hours of rain
  mu[7] <- 0.01 # cbwd - Combined wind speed

  # Prior variances to reflect the expert information
  # SampleVariance = 0.008473274
  Var0 <- 0.004 # according to sample variance
  Var[1] <- 0.1 # DEWP: No expert knowledge
  Var[2] <- 0.1 # TEMP: No expert knowledge
  Var[3] <- 0.1 # PRES: no expert knowledge
  Var[4] <- 0.1 # cbwd: No expert knowledge
  Var[5] <- 0.1 # lws: No expert knowledge
  Var[6] <- 0.1 # ls: No expert knowledge
  Var[7] <- 0.1 # lr: No expert knowledge
```

```
  # Compute corresponding prior means and variances for the standardised parameters
  muZ[1:(Nx-1)] <-  mu[1:((Nx-1))] * xsd[1:(Nx-1)] / ysd
  muZ[7] <- mu[7] # PropertType: Categorical variable not standardised
  muZ0 <- (mu0 + (sum( mu[1:(Nx-1)] * xm[1:(Nx-1)] / xsd[1:(Nx-1)] ) + mu[7])*ysd - ym) / ysd

  # Compute corresponding prior variances and variances for the standardised parameters
  VarZ[1:(Nx-1)] <- Var[1:(Nx-1)] * ( xsd[1:(Nx-1)]/ ysd )^2
  VarZ[7] <- Var[7] # PropertyType: Categorical variable is not standardised
  VarZ0 <- Var0 / (ysd^2)

}
# Specify the model for standardized data:
model {
  for ( i in 1:Ntotal ) {
    zy[i] ~ dnorm( zbeta0 + sum( zbeta[1:(Nx-1)] * zx[i,1:(Nx-1)] ) + zbeta[7]*x[i,7] , 1/zsigma^2 )
  }

  # Priors vague on standardized scale:
  zbeta0 ~ dnorm( muZ0 , 1/VarZ0 )
  for ( j in 1:Nx ) {
    zbeta[j] ~ dnorm( muZ[j] , 1/VarZ[j] )
  }

  zsigma ~ dgamma(0.001,0.01) # Standardised sigma distribution

  # Transform to original scale:
  beta[1:(Nx-1)] <- ( zbeta[1:(Nx-1)] / xsd[1:(Nx-1)] )*ysd
  beta[7] <- zbeta[7] # As cbwd was not standardised
  beta0 <- zbeta0*ysd  + ym - (sum( zbeta[1:(Nx-1)] * xm[1:(Nx-1)] / xsd[1:(Nx-1)] )+zbeta[7])*ysd
  sigma <- zsigma*ysd

  # Compute predictions at every step of the MCMC
  pred[1:2] <- beta0 + beta[1] * xPred[1:2,1] + beta[2] * xPred[1:2,2] +
               beta[4] * xPred[1:2,4] + beta[5] * xPred[1:2,5] + beta[6] * xPred[1:2,6]

  # + beta[3] * xPred[1:2,3] + beta[7] * xPred[1:2,7]

}
" # close quote for modelString
```

As we have no prior information, we have defined same prior values for the beta parameters. Also, as cbwd is a categorical variable, it was not standardised in the model definition.

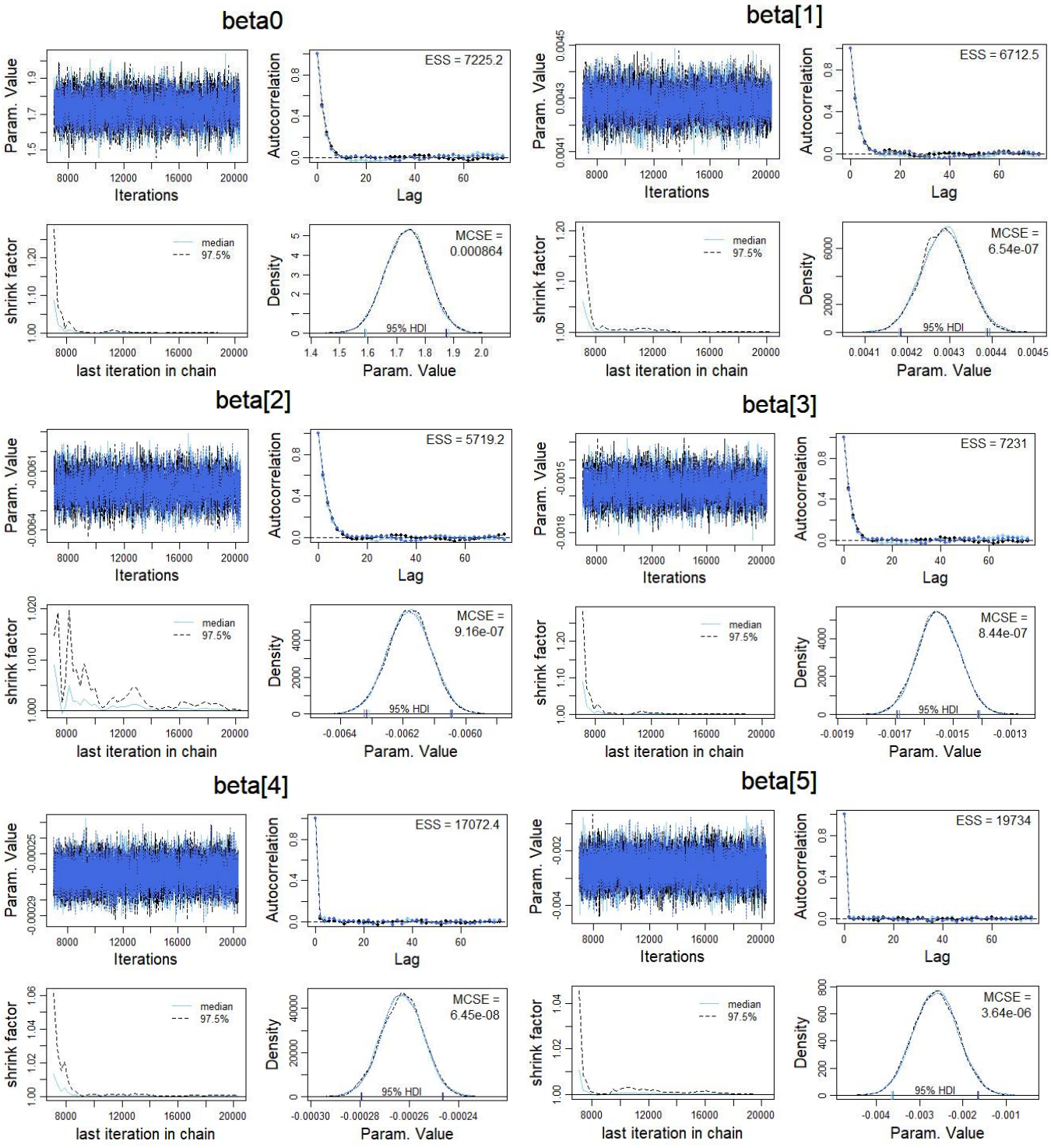
Settings:

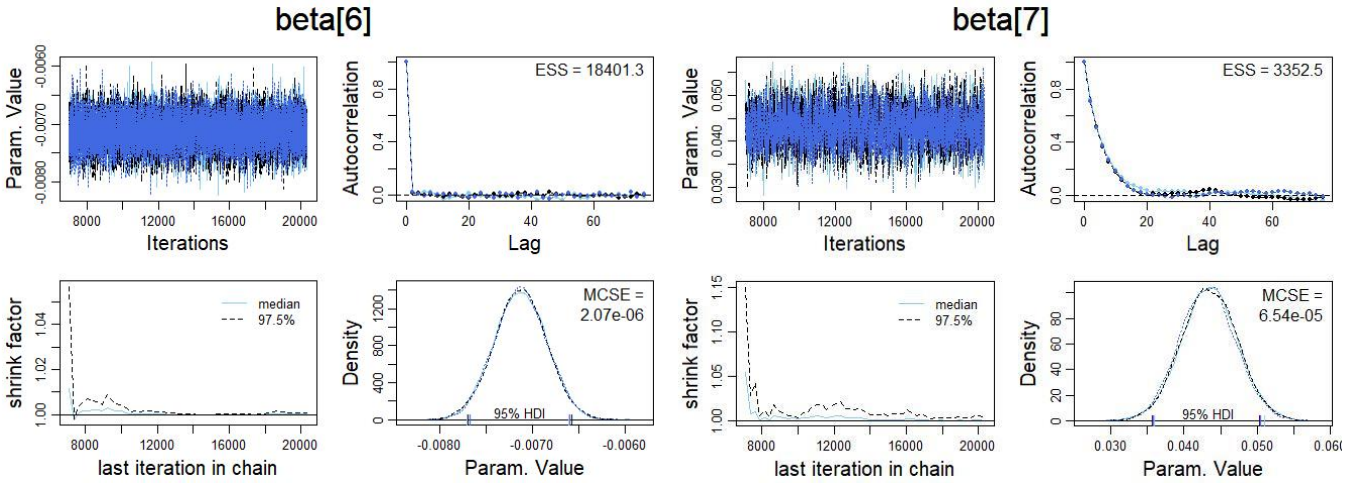
- Adaptation Steps = 2,000
- Burn-in Steps = 5,000
- Number of saved steps = 20,000
- Thinning Steps = 2

NOTE: *As the target variable is taken in thousands, all the expert information is converted to the same scale. Also, the mu and Var values are taken in thousands.*

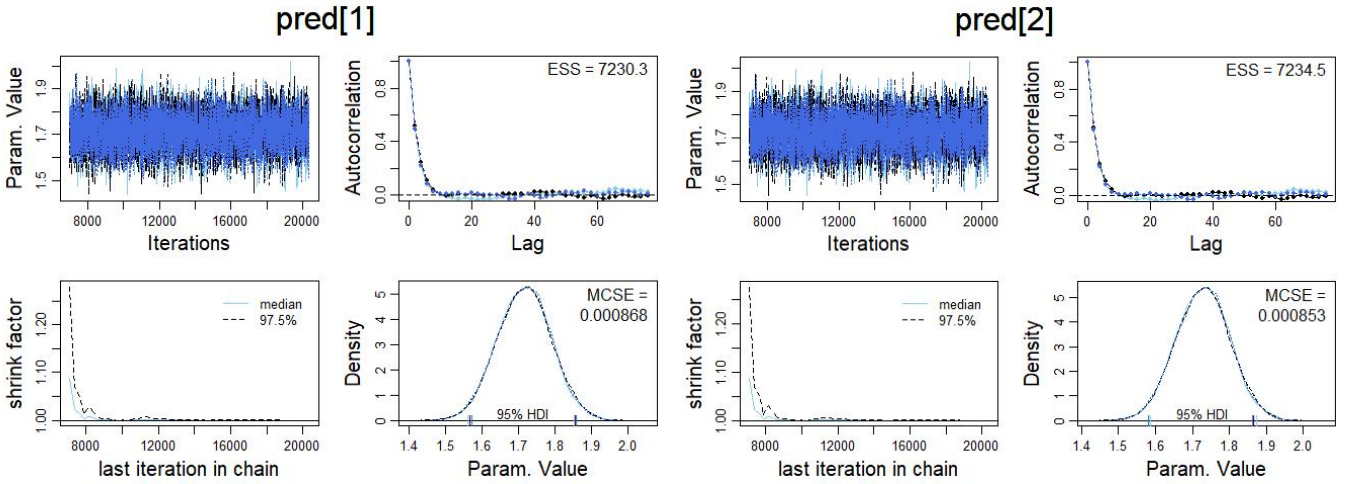
Diagnostics

Beta Parameters

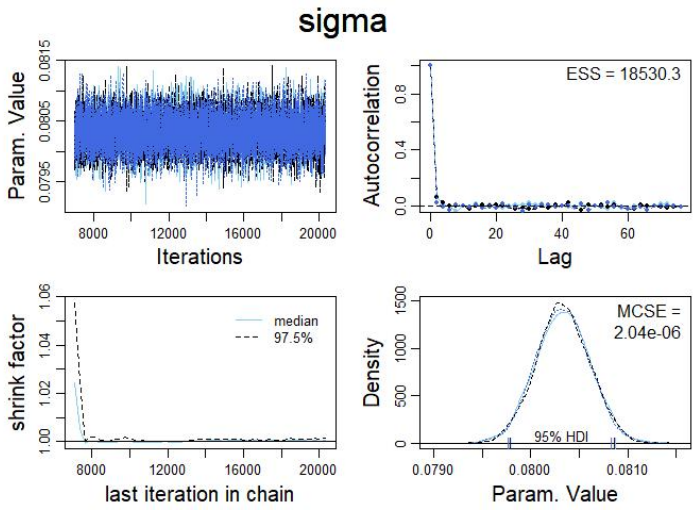




Prediction Values



Sigma



Observations of the plots

As per the diagnostic plots shown above:

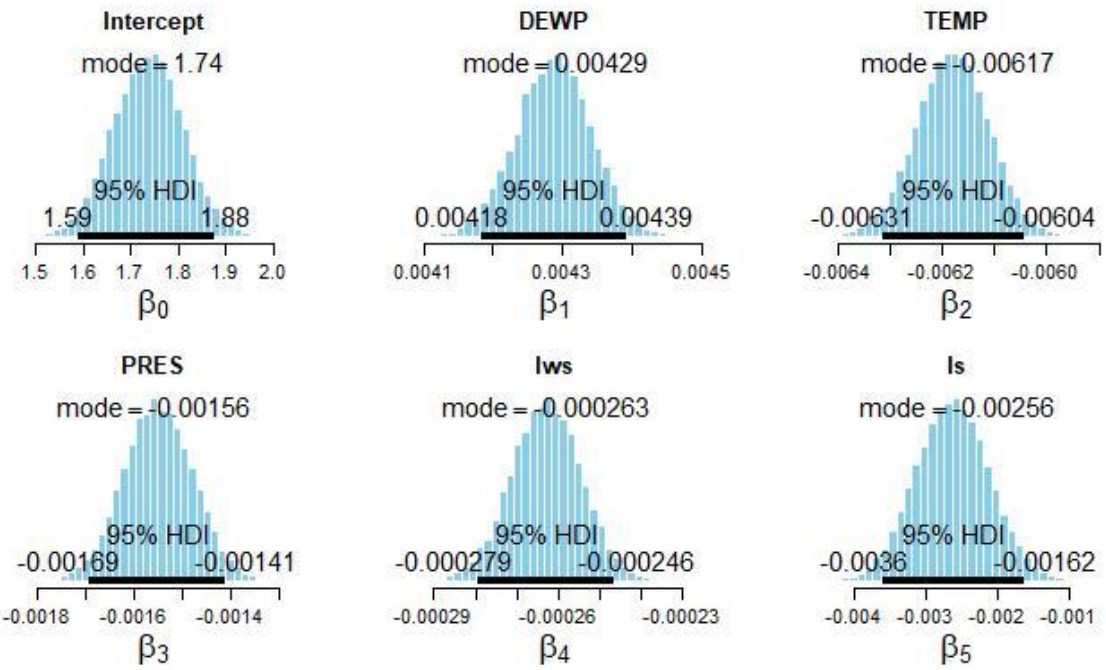
- Top-Left Plot: The chains appear to be properly overlapping and cycle around the same mean value which presents that these represent the associated pdf

- Top-Right Plot: There is no significant auto-correlation between the 4 chains and we conclude that the parameter values at successive steps in the chain are providing independent information about the posterior distribution.
- Effective Sample Size (ESS): Significantly large i.e. how much independent information there is in auto-correlated chains.
- Bottom-Left Plot: The shrink factor for both parameters is less than 1.1 which presents that the variance between chains approaches to the variance within chains after certain number of iterations
- Bottom-Right Plot: This presents the density plots of the parameter values. After the burn-in period, all chains are overlapping, which shows that chains are producing representative values from the posterior distribution.
- MCSE: Low value represents that the posterior appears to be estimated very stably.

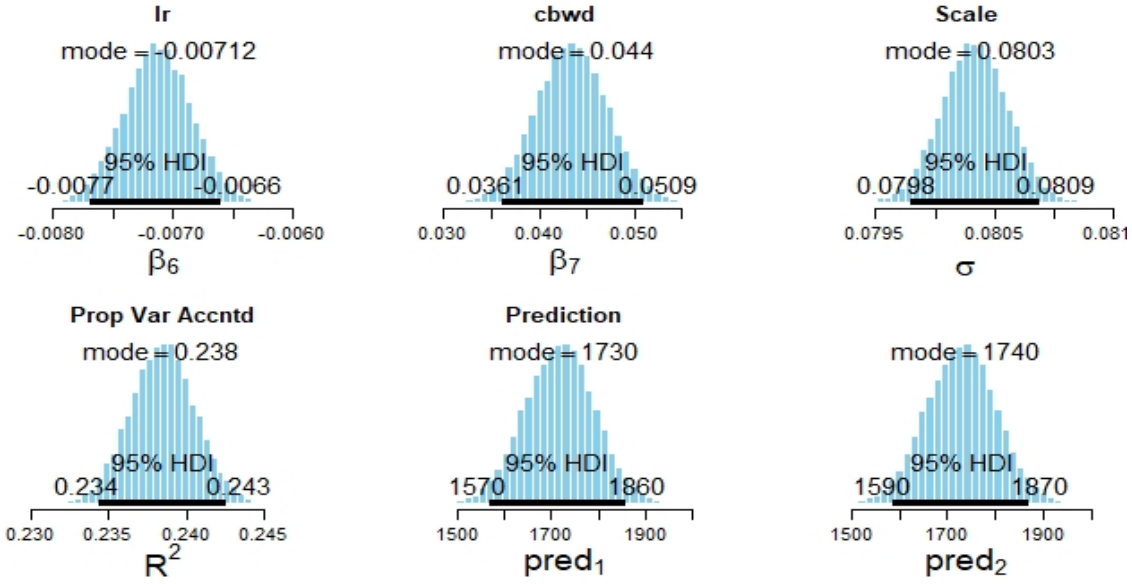
NOTE: As we increase the number of iterations, we will obtain higher ESS values.

Posterior Distribution

Beta Parameters



Prediction Values, Sigma and R^2

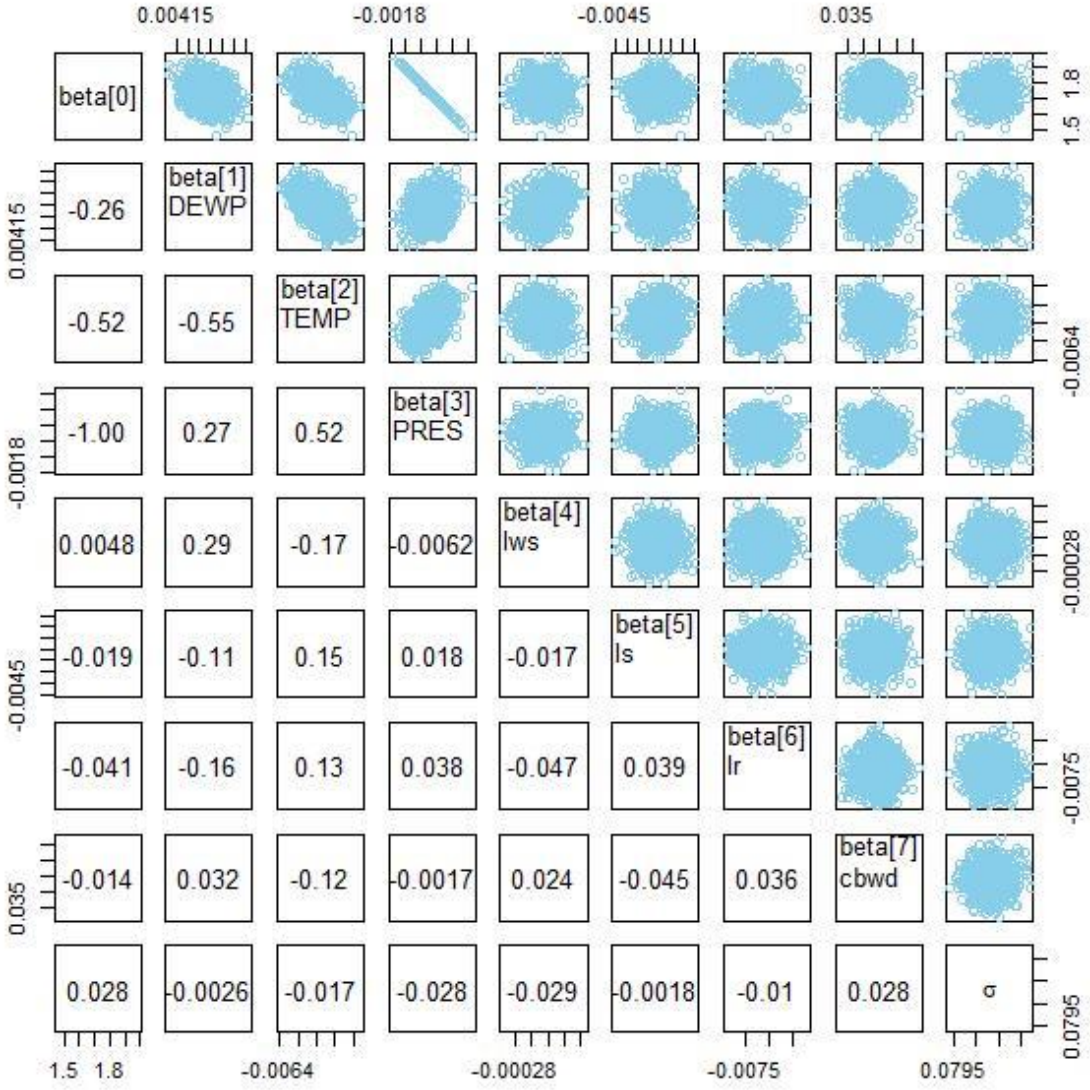


The R^2 value represents how much variance our model can explain. Low value does not mean a bad model, it helps us to conclude that more predictors are required in the model.

The predictions obtained through Bayesian analysis for given parameters are:

DEWP	TEMP	PRES	Iws	Is	Ir	cbwd	Pm2.5
-11	-5	1025	1.57	1	0	-2	**1730**
20	15	1000	2.5	0	0	1	**1740**

Pairs Plot



This pair plot is used to present the correlation between each parameter of our regression model.

There is high negative correlation between beta0 and PRES, small negative correlation between TEMP and DEWP and small positive correlation between PRES and TEMP

All other correlations represent either weak or no linear relation between them.

Conclusion

The above shows the distributions of various predictors with 95% confidence intervals. Also, the modes are displayed which are used as a measure of central tendency in Bayesian statistics.

Adjusted r squared interval is between .234 and .243. We saw the through classical methods, the value was ~0.12 while with Bayesian the value almost doubled. The reason is that even if you use non-info priors you still put some more information on the analysis this increases the explained proportion of the total variation, so you get higher R-square.

Appendix

Project Execution Code

```
# MATH2269_Applied Bayesian Statistics

# Final Assignment: Execution Script

#--------------------------------Data Read---------------------------------


beijingpm = read.csv("~/PRSA_data_2010.1.1-2014.12.31.csv")


#-------------------------Data Cleaning-------------------------------


# Removing missing values as they are only 4.7%


beijingpm <- beijingpm[!(is.na(beijingpm$pm2.5)),]


# Time variables - Year, Month and Hour are not required

# Also, S.No is removed from the dataset


beijingpm <- beijingpm[,6:13]


## Reorder the columns in the dataset


beijingpm <- beijingpm[,c(2,3,4,6,7,8,5,1)]


#-------------------------------------Data Transformation-----------------------------------


# We will set the levels of the categorical variable as -2,-1,1,2

# This step is required as the variables of linear regression require numerical values


beijingpm$cbwd <- as.character(beijingpm$cbwd)

beijingpm[beijingpm$cbwd=='NW',]$cbwd <- -2

beijingpm[beijingpm$cbwd=='cv',]$cbwd <- -1
```

```
beijingpm[beijingpm$cbwd=='NE',]$cbwd <- 1
beijingpm[beijingpm$cbwd=='SE',]$cbwd <- 2

beijingpm$cbwd <- as.factor(beijingpm$cbwd)

#--------------------------------------Specification for MCMC-------------------------------------

# beijingpm = beijingpm[sample(nrow(beijingpm),500),] # Sample was taken to test the chains

yName = "pm2.5" ; xName = c("DEWP", "TEMP",          "PRES", "Iws", "Is", "Ir","cbwd")
fileNameRoot = "Project"
numSavedSteps = 20000 ; thinSteps=2

graphFileType = "jpg"

##---------------------------------Packages & Source Reqruired---------------------------------------

# Load the relevant model into R's working memory:

source("~/Bayesian_Project_Model.r")

##---------------------------------------Generating Chains-----------------------------------------

# Generate the MCMC chain:
startTime = proc.time()
xPred = matrix(c(-11,20,-5,15,1025,1000,1.57,2.5,1,0,0,0,-2,1), nrow = 2, ncol = 7)
colnames(xPred) = c("beta[1]","beta[2]","beta[3]","beta[4]","beta[5]","beta[6]","beta[7]")
mcmcCoda = genMCMC( data=beijingpm , xName=xName , yName=yName ,
                    numSavedSteps=numSavedSteps , thinSteps=thinSteps ,
                    saveName=fileNameRoot , xPred = xPred )
```

```
stopTime = proc.time()
duration = stopTime - startTime
show(duration) # To display the duration of the process

##-----------------------------------Display Diagnostics--------------------------------------

# Display diagnostics of chain, for specified parameters:
parameterNames = varnames(mcmcCoda) # get all parameter names
for ( parName in parameterNames ) {
  diagMCMC( codaObject=mcmcCoda , parName=parName ,
            saveName=fileNameRoot , saveType=graphFileType )
}

##-----------------------------------Summary Statistics--------------------------------------

# Get summary statistics of chain:

summaryInfo = smryMCMC( mcmcCoda ,
                        saveName=fileNameRoot  )
show(summaryInfo)

##---------------------------------Display posterior information---------------------------------

plotMCMC( mcmcCoda , data=beijingpm , xName=xName , yName=yName ,
          pairsPlot=TRUE , showCurve=FALSE ,
          saveName=fileNameRoot , saveType="jpg" )

####################################--------END-------#################################
```

Model Code

```
# MATH2269_Applied Bayesian Statistics
# Final Project: Script specifying the model
```

```
#------------------------------Source File and Packages------------------------------

source("~/DBDA2E-utilities.r") # Given by Dr. Haydar Demirhan

#------------------------------Function to Generate Chain------------------------------

genMCMC = function( data , xName="x" , yName="y" ,
                    numSavedSteps=10000 , thinSteps=1 , saveName=NULL  ,
                    runjagsMethod=runjagsMethodDefault ,
                    nChains=nChainsDefault , xPred = xPred) {
 require(runjags)

 #---------------------------------The Data-------------------------------------------

 y = data[,yName]/1000
 x = data.matrix(data[,xName])
 # Do some checking that data make sense:
 if ( any( !is.finite(y) ) ) { stop("All y values must be finite.") }
 if ( any( !is.finite(x) ) ) { stop("All x values must be finite.") }
 # for (i in nrow(x)){
 #   if ( ( !is.finite(x[i]) ) ) { stop("All x values must be finite.") }
 # }
 cat("\nCORRELATION MATRIX OF PREDICTORS:\n ")
 show( round(cor(x),3) )
 cat("\n")
 flush.console()
 # Specify the data in a list, for later shipment to JAGS:
 dataList = list(
  x = x ,
  y = y ,
  Nx = dim(x)[2] ,
  Ntotal = dim(x)[1] ,
  xPred = xPred # Data for predictions
 )

 #---------------------------------------------The Model-------------------------------------

 modelString = "
 # Standardize the data:
 data {
  ym <- mean(y)
  ysd <- sd(y)
  for ( i in 1:Ntotal ) {
    zy[i] <- ( y[i] - ym ) / ysd
  }
  for ( j in 1:((Nx-1)) ) {
    xm[j] <- mean(x[,j])
    xsd[j] <- sd(x[,j])
```

```
    for ( i in 1:Ntotal ) {
      zx[i,j] <- ( x[i,j] - xm[j] ) / xsd[j]
    }
  }

  # Specify the priors for original beta parameters
  # Prior locations to reflect the expert information
  # pm2.5 in thousands taken for the model to execute
  # Sample mean = 0.09861321
  mu0 <- ym # Set to overall mean a priori based on the interpretation of constant term in
regression
  mu[1] <- 0.01 # DEWP - Dew Point
  mu[2] <- 0.01 # TEMP - Temperature
  mu[3] <- 0.01 # PRES - Pressure
  mu[4] <- 0.01 # lws - Cumulated Wind Speed
  mu[5] <- 0.01 # ls - Cumulated Hours of Snow
  mu[6] <- 0.01 # lr - Cumulated Hours of rain
  mu[7] <- 0.01 # cbwd - Combined wind speed

  # Prior variances to reflect the expert information
  # SampleVariance = 0.008473274
  Var0 <- 0.004 # according to sample variance
  Var[1] <- 0.1 # DEWP: No expert knowledge
  Var[2] <- 0.1 # TEMP: No expert knowledge
  Var[3] <- 0.1 # PRES: no expert knowledge
  Var[4] <- 0.1 # cbwd: No expert knowledge
  Var[5] <- 0.1 # lws: No expert knowledge
  Var[6] <- 0.1 # ls: No expert knowledge
  Var[7] <- 0.1 # lr: No expert knowledge


  # Compute corresponding prior means and variances for the standardised parameters
  muZ[1:(Nx-1)] <-  mu[1:((Nx-1))] * xsd[1:(Nx-1)] / ysd
  muZ[7] <- mu[7] # PropertType: Categorical variable not standardised
  muZ0 <- (mu0 + (sum( mu[1:(Nx-1)] * xm[1:(Nx-1)] / xsd[1:(Nx-1)] ) + mu[7])*ysd - ym) / ysd

  # Compute corresponding prior variances and variances for the standardised parameters
  VarZ[1:(Nx-1)] <- Var[1:(Nx-1)] * ( xsd[1:(Nx-1)]/ ysd )^2
  VarZ[7] <- Var[7] # PropertyType: Categorical variable is not standardised
  VarZ0 <- Var0 / (ysd^2)

 }
 # Specify the model for standardized data:
 model {
  for ( i in 1:Ntotal ) {
   zy[i] ~ dnorm( zbeta0 + sum( zbeta[1:(Nx-1)] * zx[i,1:(Nx-1)] ) + zbeta[7]*x[i,7] , 1/zsigma^2 )
  }
```

```
  # Priors vague on standardized scale:
  zbeta0 ~ dnorm( muZ0 , 1/VarZ0 )
  for ( j in 1:Nx ) {
    zbeta[j] ~ dnorm( muZ[j] , 1/VarZ[j] )
  }

  zsigma ~ dgamma(0.001,0.01) # Standardised sigma distribution

  # Transform to original scale:
  beta[1:(Nx-1)] <- ( zbeta[1:(Nx-1)] / xsd[1:(Nx-1)] )*ysd
  beta[7] <- zbeta[7] # As cbwd was not standardised
  beta0 <- zbeta0*ysd  + ym - (sum( zbeta[1:(Nx-1)] * xm[1:(Nx-1)] / xsd[1:(Nx-1)] )+zbeta[7])*ysd
  sigma <- zsigma*ysd

  # Compute predictions at every step of the MCMC
  pred[1:2] <- beta0 + beta[1] * xPred[1:2,1] + beta[2] * xPred[1:2,2] +
            beta[4] * xPred[1:2,4] + beta[5] * xPred[1:2,5] + beta[6] * xPred[1:2,6]

  # + beta[3] * xPred[1:2,3] + beta[7] * xPred[1:2,7]

}
" # close quote for modelString
# Write out modelString to a text file
writeLines( modelString , con="TEMPmodel.txt" )
#------------------------------------------------------------------------------

#------------------------------------------------------------------------------
# RUN THE CHAINS
parameters = c( "beta0" ,  "beta" ,  "sigma",
              "zbeta0" , "zbeta" , "zsigma" , "pred" )
adaptSteps = 2000  # Number of steps to "tune" the samplers
burnInSteps = 5000
runJagsOut <- run.jags( method=runjagsMethod ,
                    model="TEMPmodel.txt" ,
                    monitor=parameters ,
                    data=dataList ,
                    n.chains=nChains ,
                    adapt=adaptSteps ,
                    burnin=burnInSteps ,
                    sample=ceiling(numSavedSteps/nChains) ,
                    thin=thinSteps ,
                    summarise=FALSE ,
                    plots=FALSE )
codaSamples = as.mcmc.list( runJagsOut )

if ( !is.null(saveName) ) {
  save( codaSamples , file=paste(saveName,"Mcmc.Rdata",sep="") )
}
```

```
  return( codaSamples )
} # end function

#===============================================================================

smryMCMC = function(  codaSamples ,
                      saveName=NULL) {
  summaryInfo = NULL
  mcmcMat = as.matrix(codaSamples,chains=TRUE)
  paramName = colnames(mcmcMat)
  for ( pName in paramName ) {
    summaryInfo = rbind( summaryInfo , summarizePost( mcmcMat[,pName] ) )
  }
  rownames(summaryInfo) = paramName
  # summaryInfo = rbind( summaryInfo ,
  #                      "log10(nu)" = summarizePost( log10(mcmcMat[,"nu"]) ) )
  if ( !is.null(saveName) ) {
    write.csv( summaryInfo , file=paste(saveName,"SummaryInfo.csv",sep="") )
  }

  return( summaryInfo)
}

#===============================================================================

plotMCMC = function( codaSamples , data , xName="x" , yName="y" ,
                     showCurve=FALSE ,  pairsPlot=FALSE ,
                     saveName=NULL , saveType="jpg" ) {
  # showCurve is TRUE or FALSE and indicates whether the posterior should
  #   be displayed as a histogram (by default) or by an approximate curve.
  # pairsPlot is TRUE or FALSE and indicates whether scatterplots of pairs
  #   of parameters should be displayed.
  #-----------------------------------------------------------------------
  y = data[,yName]/1000
  x = data.matrix(data[,xName])
  mcmcMat = as.matrix(codaSamples,chains=TRUE)
  chainLength = NROW( mcmcMat )
  zbeta0 = mcmcMat[,"zbeta0"]
  zbeta  = mcmcMat[,grep("^zbeta$|^zbeta\\[",colnames(mcmcMat))]
  if ( ncol(x)==1 ) { zbeta = matrix( zbeta , ncol=1 ) }
  zsigma = mcmcMat[,"zsigma"]
  beta0 = mcmcMat[,"beta0"]
  beta  = mcmcMat[,grep("^beta$|^beta\\[",colnames(mcmcMat))]
  if ( ncol(x)==1 ) { beta = matrix( beta , ncol=1 ) }
  sigma = mcmcMat[,"sigma"]
  pred  = mcmcMat[,grep("^pred$|^pred\\[",colnames(mcmcMat))] # Added by Vishesh
  pred = pred*1000 # To be checked, converting back to the given pm2.5 values
```

```
#-------------------------------------------------------------------------
# Compute R^2 for credible parameters:
YcorX = cor( y , x ) # correlation of y with each x predictor
Rsq = zbeta %*% matrix( YcorX , ncol=1 )
#-------------------------------------------------------------------------
if ( pairsPlot ) {
  # Plot the parameters pairwise, to see correlations:
  openGraph()
  nPtToPlot = 1000
  plotIdx = floor(seq(1,chainLength,by=chainLength/nPtToPlot))
  panel.cor = function(x, y, digits=2, prefix="", cex.cor, ...) {
    usr = par("usr"); on.exit(par(usr))
    par(usr = c(0, 1, 0, 1))
    r = (cor(x, y))
    txt = format(c(r, 0.123456789), digits=digits)[1]
    txt = paste(prefix, txt, sep="")
    if(missing(cex.cor)) cex.cor <- 0.8/strwidth(txt)
    text(0.5, 0.5, txt, cex=1.25 ) # was cex=cex.cor*r
  }
  pairs( cbind( beta0 , beta , sigma )[plotIdx,] ,
        labels=c( "beta[0]" ,
                  paste0("beta[",1:ncol(beta),"]\n",xName),
                  expression(sigma) ) ,
        lower.panel=panel.cor , col="skyblue" )
  if ( !is.null(saveName) ) {
    saveGraph( file=paste(saveName,"PostPairs",sep=""), type=saveType)
  }
}
#-------------------------------------------------------------------------
# Marginal histograms:

decideOpenGraph = function( panelCount , saveName , finished=FALSE ,
                            nRow=2 , nCol=3 ) {
  # If finishing a set:
  if ( finished==TRUE ) {
    if ( !is.null(saveName) ) {
      saveGraph( file=paste0(saveName,ceiling((panelCount-1)/(nRow*nCol))),
                 type=saveType)
    }
    panelCount = 1 # re-set panelCount
    return(panelCount)
  } else {
  # If this is first panel of a graph:
  if ( ( panelCount %% (nRow*nCol) ) == 1 ) {
    # If previous graph was open, save previous one:
    if ( panelCount>1 & !is.null(saveName) ) {
      saveGraph( file=paste0(saveName,(panelCount%/%(nRow*nCol))),
                 type=saveType)
```

```
    }
    # Open new graph
    openGraph(width=nCol*7.0/3,height=nRow*2.0)
    layout( matrix( 1:(nRow*nCol) , nrow=nRow, byrow=TRUE ) )
    par( mar=c(4,4,2.5,0.5) , mgp=c(2.5,0.7,0) )
  }
  # Increment and return panel count:
  panelCount = panelCount+1
  return(panelCount)
  }
}

# Original scale:
panelCount = 1
panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMarg") )
histInfo = plotPost( beta0 , cex.lab = 1.75 , showCurve=showCurve ,
                xlab=bquote(beta[0]) , main="Intercept" )
for ( bIdx in 1:ncol(beta) ) {
  panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMarg") )
  histInfo = plotPost( beta[,bIdx] , cex.lab = 1.75 , showCurve=showCurve ,
                  xlab=bquote(beta[.(bIdx)]) , main=xName[bIdx] )
}
panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMarg") )
histInfo = plotPost( sigma , cex.lab = 1.75 , showCurve=showCurve ,
                xlab=bquote(sigma) , main=paste("Scale") )
panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMarg") )
histInfo = plotPost( Rsq , cex.lab = 1.75 , showCurve=showCurve ,
                xlab=bquote(R^2) , main=paste("Prop Var Accntd") )
for ( pIdx in 1:ncol(pred) ) {
panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMarg") )
histInfo = plotPost( pred[,pIdx] , cex.lab = 1.75 , showCurve=showCurve ,
                xlab=bquote(pred[.(pIdx)]) , main="Prediction"[pIdx])
} # Modified by Vishesh #, finished=TRUE : Removed
# Standardized scale:
panelCount = 1
panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMargZ") )
histInfo = plotPost( zbeta0 , cex.lab = 1.75 , showCurve=showCurve ,
                xlab=bquote(z*beta[0]) , main="Intercept" )
for ( bIdx in 1:(ncol(beta)-1) ) {
  panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMargZ") )
  histInfo = plotPost( zbeta[,bIdx] , cex.lab = 1.75 , showCurve=showCurve ,
                  xlab=bquote(z*beta[.(bIdx)]) , main=xName[bIdx] )
}
panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMargZ") )
histInfo = plotPost( zsigma , cex.lab = 1.75 , showCurve=showCurve ,
                xlab=bquote(z*sigma) , main=paste("Scale") )
panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMargZ") )
histInfo = plotPost( Rsq , cex.lab = 1.75 , showCurve=showCurve ,
```

```
                    xlab=bquote(R^2) , main=paste("Prop Var Accntd") )
  panelCount = decideOpenGraph( panelCount , finished=TRUE ,
saveName=paste0(saveName,"PostMargZ") )

  #------------------------------------------------------------------------
}
#=============================================================================
```