MATH2305 Applied Bayesian Statistics - Assignment 2
Jason Thomas - s3907634

Table of Contents

1. Introduction

Australia's real estate prices are a concern for everyone who watches the economy, and anyone who wishes to purchase a house. This report will seek to estimate a property's selling price based on:

- 10,000 observed property sales in Melbourne Victoria.
- Expert information

To make use of these two sources of information, this report uses Bayesian Regression. This is achievable with these tools: R and JAGS.

1.1 Report Objectives

This report seeks to address the following

- Q1: Create a JAGS model diagram showing the multiple linear regression setting in this problem.
- Q2: Specify the prior distributions reflecting the expert information for each predictor.
- Q3: Create JAGS data and model blocks based on the model diagram and prior distributions at the previous steps.
- Q4: Compile your model and create Markov chains using the compiled model.
- Q5: Assess the appropriateness of the chains for each parameter using the MCMC diagnostics.
- Q6: Display the posterior distribution of each parameter and draw inferences on Bayesian point and interval estimates.
- Q7: Use the Bayesian point estimates of the model parameters to write the predictions model.
- Q8: Find the predictions of sale prices for the properties given below

Property No	**Area**	**Bedrooms**	**Bathrooms**	**CarParks**	**PropertyType**
1	600	2	2	1	Unit
2	800	3	1	2	House
3	1500	2	1	1	House
4	2500	5	4	4	House
5	250	3	2	1	Unit

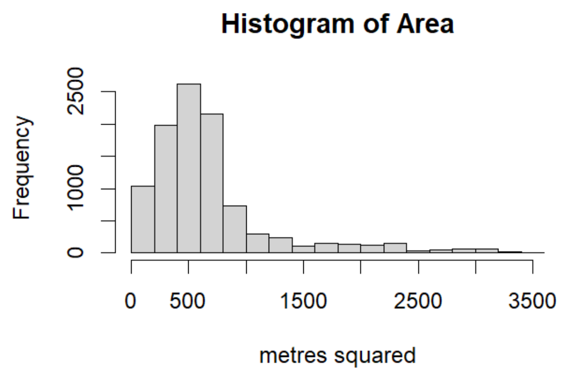
1.2 Data

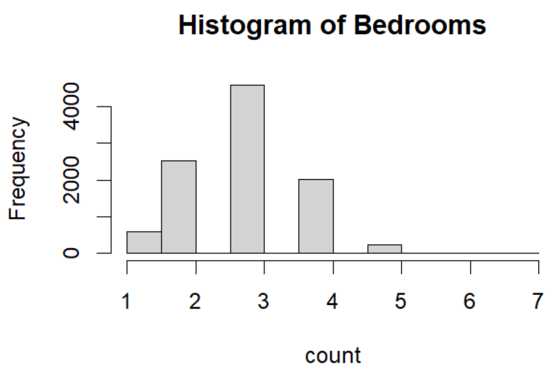
The dataset contains the details and final selling price of 10,000 properties.

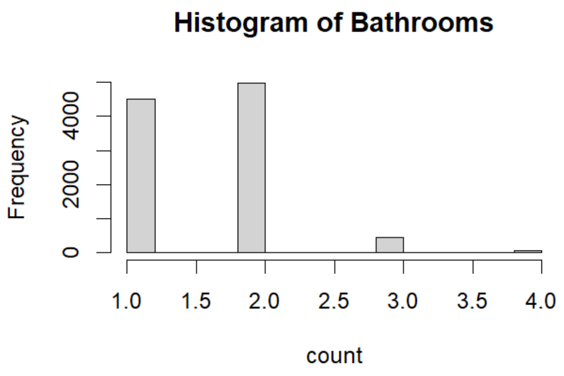
The variables of interest are:

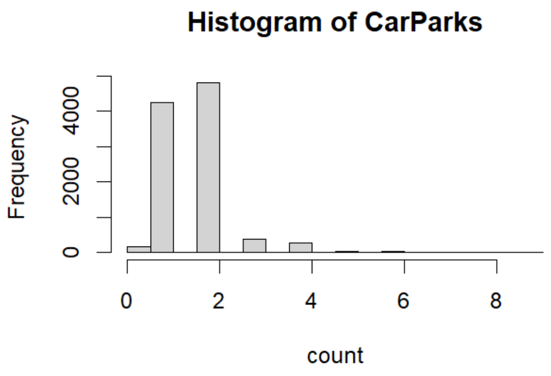
- Sale Price ($100,000s)
- Area (m^2)
- Bedrooms (count)
- Bathrooms (count)
- CarParks (count)
- PropertyType (categorical): 1 for a unit, 0 for a house

The independent variables have these distributions:



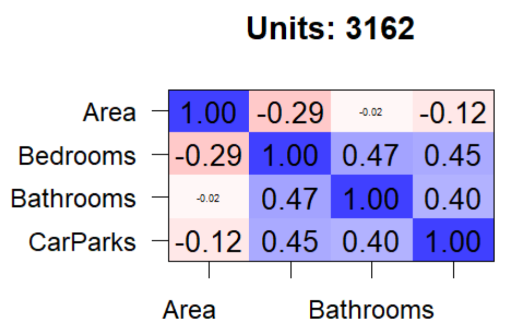


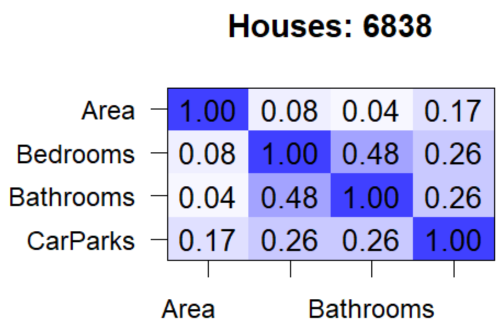




These variables appear to follow normal-like, although skewed, distributions.

There are these correlations between the non-categorical data:





Then, for units and houses, more area seems to have no strong relationship with Bedrooms, Bathrooms or CarParks. This seems counterintuitive. For houses in particular, there seems to be some variability: houses might have many bedrooms and bathrooms, but fewer carparks, or many carparks but fewer bathrooms and bedrooms.

1.3 Expert information

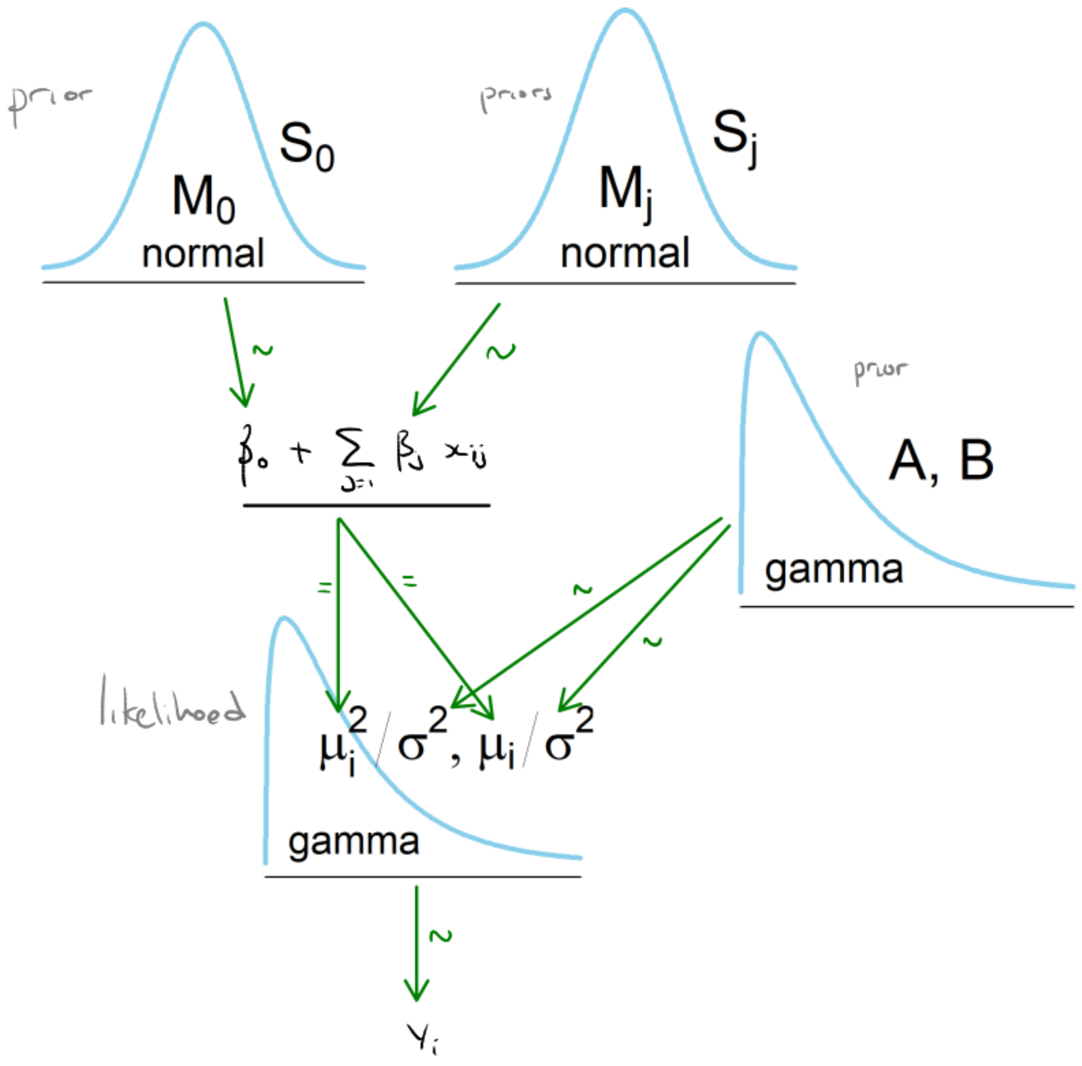
To conduct this analysis, we sought the opinion of an expert in Australia's property sector.

We obtained this information:

	Expert knowledge	Expert Confidence
Area	$90 increase per m^2	Very strong
Bedrooms	$100,000 increase per bedroom	Weak
Bathrooms	No information	N/A
CarParks	$120,000 increase per car park	Strong
PropertyType	$150,000 decrease for units	Very strong

2. Analysis

2.1 (Q1)



prior
M_0 normal S_0
priors
M_j normal S_j
prior
A, B gamma
$\beta_0 + \sum_{j=1} \beta_j x_{ij}$
likelihood
$\mu_i^2 / \sigma^2, \mu_i / \sigma^2$ gamma
Y_i

2.2 (Q2)

Regression parameter priors:

In the model diagram, the top two prior distributions reflect that regression parameters can be positive or negative real numbers. This means that normal priors are acceptable.

There is no information for the y intercept.

	Expert knowledge	Confidence	Variable type	Var for Normal
Area	90	Very strong	Discrete	0.01
Bedrooms	100,000	Weak	Discrete	1
Bathrooms	N/A	N/A	Discrete	10
CarParks	120,000	Strong	Discrete	0.1
PropertyType	-150,000	Very strong	Categorical	0.01

These variances reflect the uncertainty that the expert has. In the case of bathrooms, 10 is such a large variance that this is an uninformative prior.

Gamma prior

We need this prior to generate a distribution for the variance parameter for the Gamma likelihood. Variance is defined to have range of $[0, \infty]$, and is a real number. That means that a Gamma prior is a good choice.

2.3 (Q3)

Please see the code that is provided as an appendix, for the data and model blocks. It is below a comment ***MODEL AND DATA.***

2.4 (Q4)

Trial and error led to these MCMC settings:

adaptSteps = 3000

burnInSteps = 4000

nChains = 2

thinSteps = 12

numSavedSteps = 4000

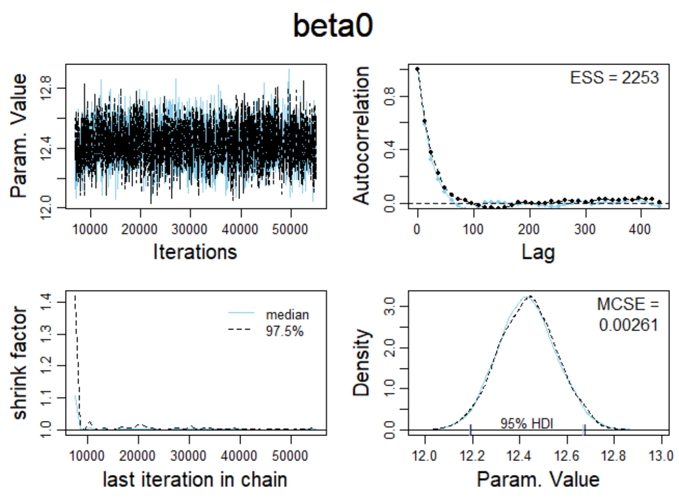
Please see the code appendix for the ***jags.model()*** function that does this.

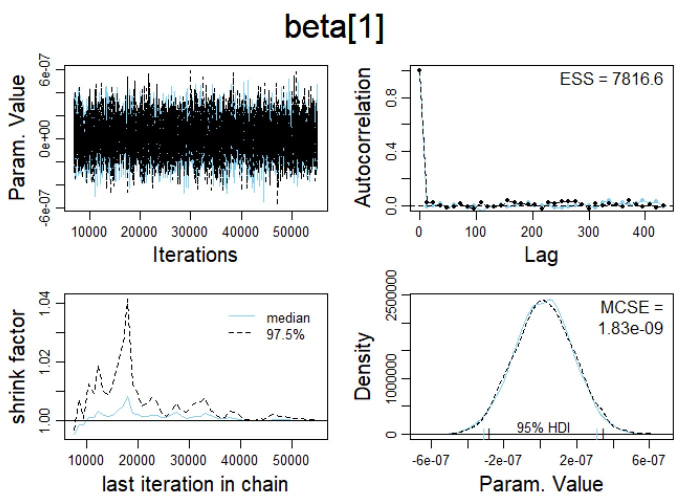
2.5 (Q5)

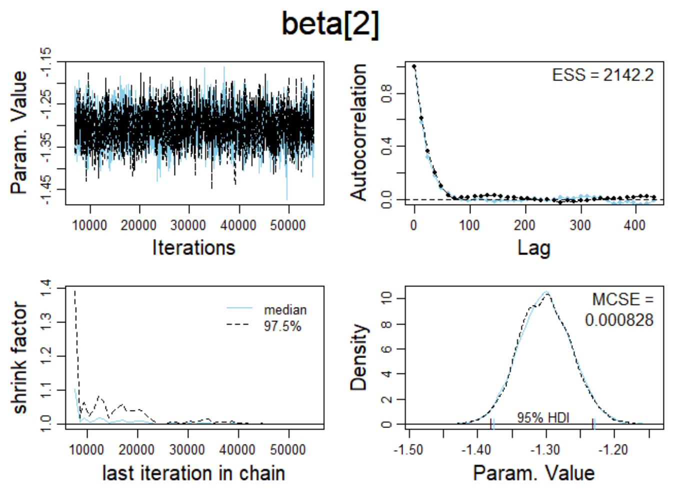
The MCMC settings in question 4 has led to favourable results for all parameters.

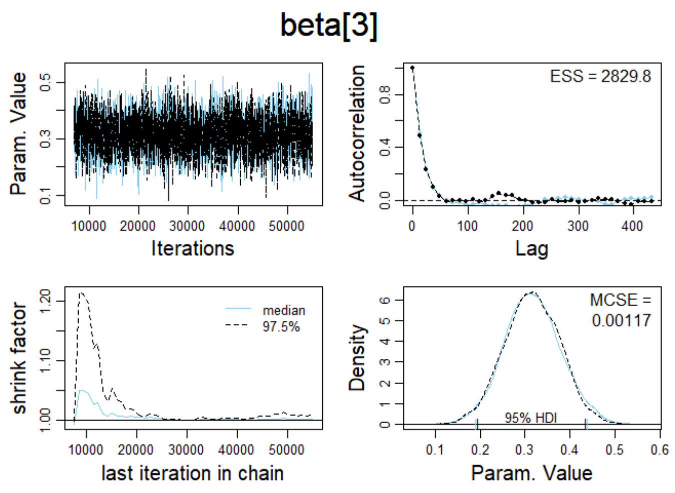
In every case:

- Representativeness: the chains overlap, so the burn-in is acceptable.
- Autocorrelation: this diminishes quickly, and ESS is high, so the amount of thinning is acceptable.
- Shrink factor: all processes converge to approximately 1.0.
- Standard error: all chains have overlapping densities, and low MCSE.

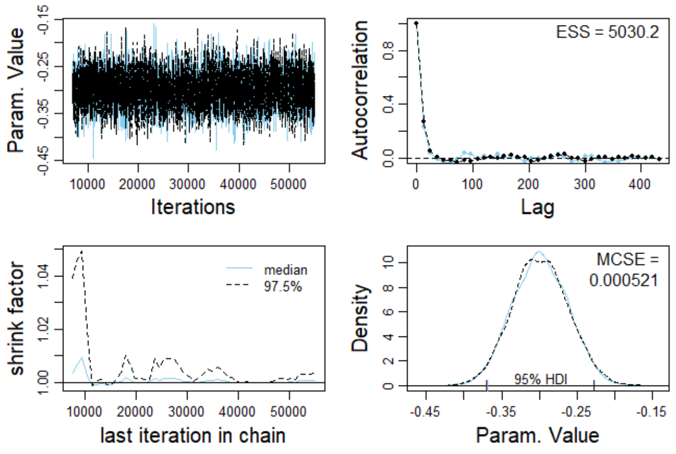




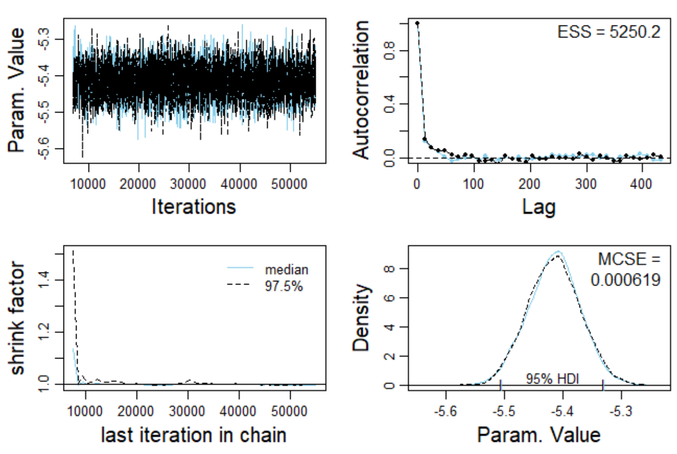




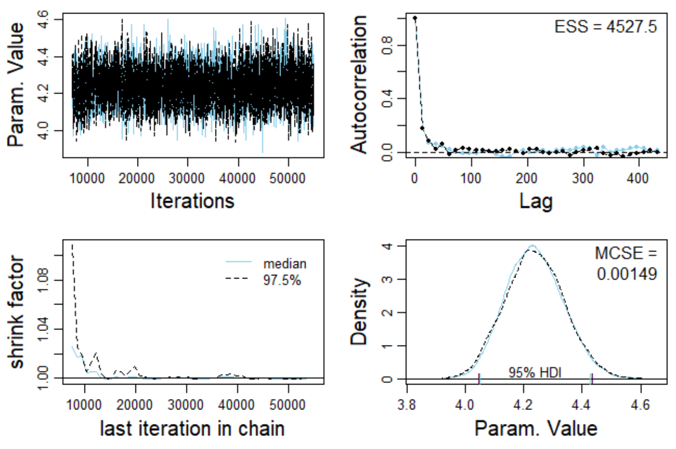
beta[4]



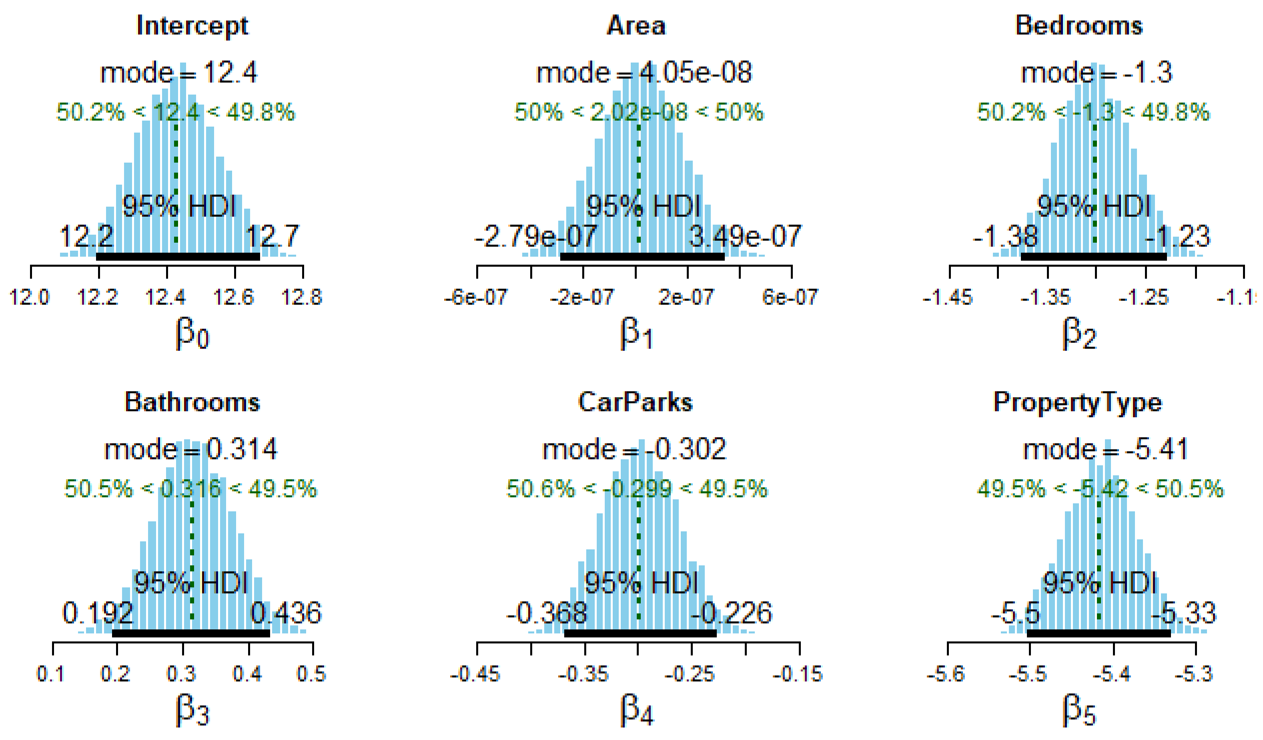
beta[5]



tau



2.6 (Q6)



These are surprising results. These results appear to not make sense, since one would not expect that an extra bedroom or carpark would decrease a property's value. But the model in the code appears to be correct. It's not clear to the author how the JAGS model could be modified to address this.

The JAGS model is as correct as the report's author could make it.

The rest of this report will use the assumption that these parameters are correct, even though they are surprising.

Then, the regression parameters are:

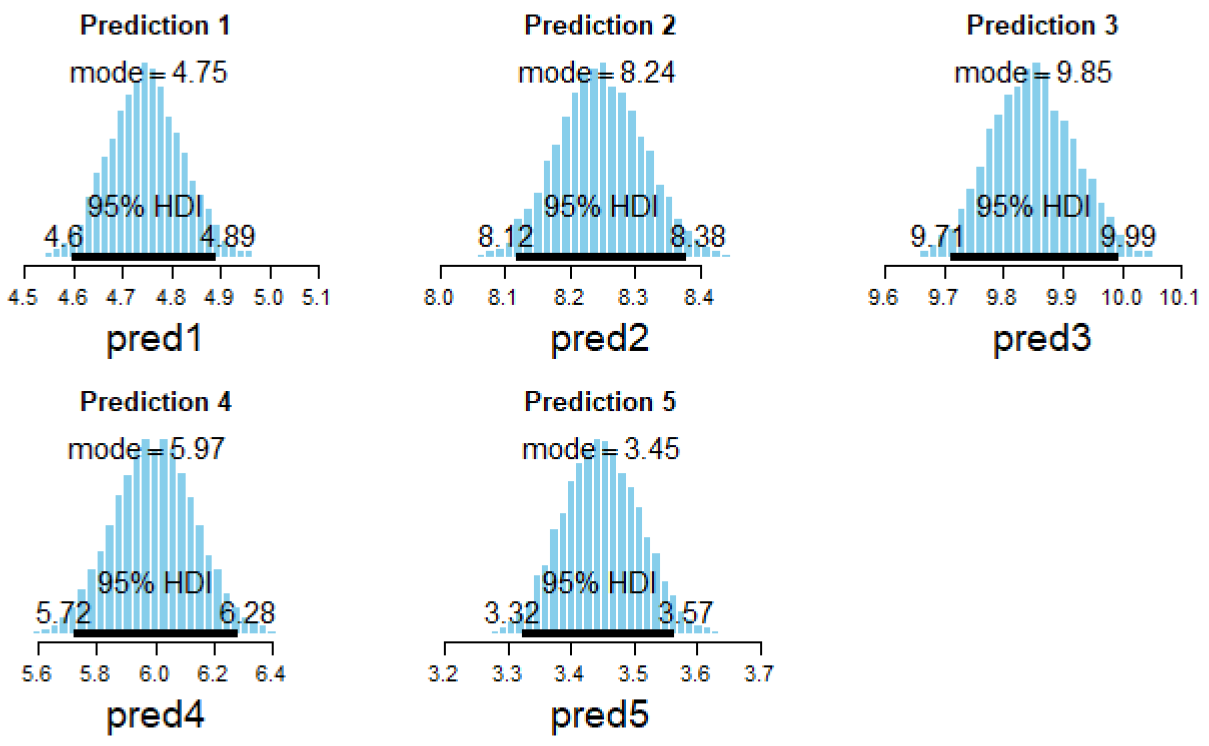
- Intercept: 12.4. This is between 12.2 and 12.7 with 0.95 probability.
- Area: 4.05e-08. This is between -2.79e-07 and 3.49e-07 with 0.95 probability.
- Bedrooms: -1.3. This is between -1.38 and -1.23 with 0.95 probability.
- Bathrooms: 0.313. This is between 0.192 and 0.436 with 0.95 probability.
- Carparks: -0.302. This is between -0.368 and -0.226 with 0.95 probability.
- PropertyType: -5.41. This is between -5.5 and -5.33 with 0.95 probability.

2.7 (Q7)

This code is implemented in the model block, of the R code in the appendix:

```
for ( i in 1:5 ) {
  pred[i] <- beta0 + beta[1] * xPred[i,1] + beta[2] * xPred[i,2] +
            beta[3] * xPred[i,3] + beta[4] * xPred[i,4] + beta[5] * xPred[i,5]
}
```

2.8 (Q8)



Each posterior here corresponds to one property given in the report introduction.

The predicted property prices are:

- Property 1: $475,000. This is between $460,000 and $489,000 with 0.95 probability.
- Property 2: $824,000. This is between $812,000 and $838,000 with 0.95 probability.
- Property 2: $985,000. This is between $971,000 and $999,000 with 0.95 probability.
- Property 2: $597,000. This is between $572,000 and $628,000 with 0.95 probability.
- Property 2: $345,000. This is between $332,000 and $357,000 with 0.95 probability.

3. Conclusion

This report sought to estimate property prices of five properties by combining empirical data with expert information, and by using Bayesian Regression, R and JAGS.

The results in 2.6 are unexpected and surprising, and indicate that the model that JAGS used could be wrong. The model was the author's best attempt at making a correct model.

The recommendation is to share the model with a more experienced analyst to see if these results are incorrect.

4. Appendix

Please copy and paste this into RStudio for ease of reading.

```
graphics.off() # This closes all of R's graphics windows.

rm(list=ls())  # Careful! This clears all of R's memory!

library(ggplot2)

library(ggpubr)

library(ks)

library(rjags)

library(runjags)

source("DBDA2E-utilities.R")


#===============PRELIMINARY FUNCTIONS FOR POSTERIOR
INFERENCES====================


smryMCMC = function(  codaSamples , compVal = NULL,  saveName=NULL) {

 summaryInfo = NULL

 mcmcMat = as.matrix(codaSamples,chains=TRUE)

 paramName = colnames(mcmcMat)

 for ( pName in paramName ) {

  if (pName %in% colnames(compVal)){

   if (!is.na(compVal[pName])) {

    summaryInfo = rbind( summaryInfo , summarizePost( paramSampleVec = mcmcMat[,pName] ,

                                                      compVal = as.numeric(compVal[pName]) ))

   }

   else {

    summaryInfo = rbind( summaryInfo , summarizePost( paramSampleVec = mcmcMat[,pName] ) )

   }

  } else {

   summaryInfo = rbind( summaryInfo , summarizePost( paramSampleVec = mcmcMat[,pName] ) )
  }
 }
 rownames(summaryInfo) = paramName

 # summaryInfo = rbind( summaryInfo ,
 #                  "tau" = summarizePost( mcmcMat[,"tau"] ) )
 if ( !is.null(saveName) ) {
  write.csv( summaryInfo , file=paste(saveName,"SummaryInfo.csv",sep="") )
 }
 return( summaryInfo )
}

#==============================================================================
plotMCMC_HD = function( codaSamples , data , xName="x" , yName="y" ,
                 showCurve=FALSE ,  pairsPlot=FALSE , compVal = NULL,
                 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]
 x = as.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 ) }
 zVar = mcmcMat[,"zVar"]
 beta0 = mcmcMat[,"beta0"]
```

```
beta  = mcmcMat[,grep("^beta$|^beta\\[",colnames(mcmcMat))]
if ( ncol(x)==1 ) { beta = matrix( beta , ncol=1 ) }
tau = mcmcMat[,"tau"]
pred1 = mcmcMat[,"pred[1]"] # Added by Demirhan
pred2 = mcmcMat[,"pred[2]"] # Added by Demirhan
pred3 = mcmcMat[,"pred[3]"] # Added by Demirhan
pred4 = mcmcMat[,"pred[4]"] # Added by Demirhan
pred5 = mcmcMat[,"pred[5]"] # Added by Demirhan
#-----------------------------------------------------------------------------
# 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 , tau )[plotIdx,] ,
      labels=c( "beta[0]" ,
             paste0("beta[",1:ncol(beta),"]\n",xName) ,
            expression(tau) ) ,
       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
if (!is.na(compVal["beta0"])){
  panelCount = decideOpenGraph( panelCount , saveName=paste0(saveName,"PostMarg") )
  histInfo = plotPost( beta0 , cex.lab = 1.75 , showCurve=showCurve ,
                  xlab=bquote(beta[0]) , main="Intercept", compVal = as.numeric(compVal["beta0"] ))
} else {
  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") )
  if (!is.na(compVal[paste0("beta[",bIdx,"]")])) {
    print(c("Plotting beta", bIdx))
    histInfo = plotPost( beta[,bIdx] , cex.lab = 1.75 , showCurve=showCurve ,
                    xlab=bquote(beta[.(bIdx)]) , main=xName[bIdx],
                    compVal = as.numeric(compVal[paste0("beta[",bIdx,"]")]))
  } else{
    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( tau , cex.lab = 1.75 , showCurve=showCurve ,
#            xlab=bquote(tau) , 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") )
panelCount = decideOpenGraph( panelCount ,  saveName=paste0(saveName,"PostMarg") )
histInfo = plotPost( pred1 , cex.lab = 1.75 , showCurve=showCurve ,
            xlab="pred1" , main="Prediction 1" ) # Added by Demirhan
panelCount = decideOpenGraph( panelCount ,  saveName=paste0(saveName,"PostMarg") )
histInfo = plotPost( pred2 , cex.lab = 1.75 , showCurve=showCurve ,
            xlab="pred2" , main="Prediction 2" ) # Added by Demirhan
panelCount = decideOpenGraph( panelCount ,  saveName=paste0(saveName,"PostMarg") )
histInfo = plotPost( pred3 , cex.lab = 1.75 , showCurve=showCurve ,
            xlab="pred3" , main="Prediction 3" ) # Added by Thomas
panelCount = decideOpenGraph( panelCount ,  saveName=paste0(saveName,"PostMarg") )
histInfo = plotPost( pred4 , cex.lab = 1.75 , showCurve=showCurve ,
            xlab="pred4" , main="Prediction 4" ) # Added by Thomas
panelCount = decideOpenGraph( panelCount ,  saveName=paste0(saveName,"PostMarg") )
histInfo = plotPost( pred5 , cex.lab = 1.75 , showCurve=showCurve ,
            xlab="pred5" , main="Prediction 5" ) # Added by Thomas

# 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) ) {
  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( zVar , cex.lab = 1.75 , showCurve=showCurve ,
                   xlab=bquote(z*tau) , 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") )

  #-----------------------------------------------------------------------------
}

#===============PRELIMINARY FUNCTIONS FOR POSTERIOR
INFERENCES====================



myData <- read.csv("Assignment2PropertyPrices.csv")
myData <- myData[sample(nrow(myData)),] # random sample
myData <- head(myData, 1000)

# THE DATA.
y = myData[,"SalePrice.100K."]
x = as.matrix(myData[,c("Area","Bedrooms","Bathrooms","CarParks","PropertyType")])

#################################################
# DESCRIPTIVE CHECKS

cat("\nCORRELATION MATRIX OF PREDICTORcS:\n ")
```

```
show( round(cor(x),3) )


unitCount = as.numeric(length(which(myData$PropertyType==1)))

houseCount = as.numeric(length(which(myData$PropertyType==0)))


library(psych)

#corPlot(x[,1:4], cex = 0.75, main="All property types")

#corPlot(x[which(myData$PropertyType==1),1:4], cex = 0.75, main=sprintf("Units: %d", unitCount))

#corPlot(x[which(myData$PropertyType==0),1:4], cex = 0.75, main=sprintf("Houses: %d",
houseCount))


cat("\n")

hist(myData$Area,

   main = "Histogram of Area",

   xlab = "metres squared")

hist(myData$Bedrooms,

   main = "Histogram of Bedrooms",

   xlab = "count")

hist(myData$Bathrooms,

   main = "Histogram of Bathrooms",

   xlab = "count")

hist(myData$CarParks,

   main = "Histogram of CarParks",

   xlab = "count")


table(myData$PropertyType)


#################################################

# SETUP BEFORE MCMC
```

```
dataList <- list(
  x = x ,
  y = y ,
  Nx = dim(x)[2] ,    # variable count
  Ntotal = dim(x)[1]  # observation count
)

# allow jags to determine initialising values, with adapt step

# THE MODEL.
modelString = "
# Standardize the data:
data {
  priorMu0   <- 1
  priorMu[1] <- 90      / 100000 # divide by 100000, since y value is scaled that way
  priorMu[2] <- 100000  / 100000
  priorMu[3] <- 1
  priorMu[4] <- 120000  / 100000
  priorMu[5] <- -150000 / 100000 # unit = 1, house = 0, unit sells $150000 less

  priorVar0   <- 1
  priorVar[1] <- 0.01 # very strong knowledge
  priorVar[2] <- 1    # weak knowledge
  priorVar[3] <- 20   # no expert knowledge
  priorVar[4] <- 0.1  # strong
  priorVar[5] <- 0.01 # very strong

  ysd <- sd(y)
  for ( i in 1:Ntotal ) {
    zy[i] <- y[i] / ysd           # standardise for use in gamma likelihood
}
for ( j in 1:Nx ) {
 xsd[j] <-   sd(x[,j])
 for ( i in 1:Ntotal ) {
   zx[i,j] <- ifelse( j == Nx , x[i,j] , x[i,j] / xsd[j] ) # last var is dummy, don't standardise it
 }

 # now standardise the prior mean and variance for this variable j
 # but not for the last variable, because it is categorical binary
 zPriorMu[j] <- ifelse( j == Nx , priorMu[j] , priorMu[j] / xsd[j] )
 zPriorVar[j] <- ifelse( j == Nx , priorVar[j] , priorVar[j] / xsd[j] )
}

# Specify the values of indepdenent variables for prediction
# There are 5 predictions, and 5 values for each (over 5 variables)
xPred[1,1] <- 600
xPred[2,1] <- 800
xPred[3,1] <- 1500
xPred[4,1] <- 2500
xPred[5,1] <- 250
xPred[1,2] <- 2
xPred[2,2] <- 3
xPred[3,2] <- 2
xPred[4,2] <- 5
xPred[5,2] <- 3
xPred[1,3] <- 2
xPred[2,3] <- 1
xPred[3,3] <- 1
xPred[4,3] <- 4
xPred[5,3] <- 2
```

```
 xPred[1,4] <- 1
 xPred[2,4] <- 2
 xPred[3,4] <- 1
 xPred[4,4] <- 4
 xPred[5,4] <- 1
 xPred[1,5] <- 1
 xPred[2,5] <- 0
 xPred[3,5] <- 0
 xPred[4,5] <- 0
 xPred[5,5] <- 1
}
# Specify the model for scaled data:
model {
 for ( i in 1:Ntotal ) {
  zy[i] ~ dgamma( (mu[i]^2)/zVar , mu[i]/zVar )    # likelihood
  mu[i] <- zbeta0 + sum( zbeta[1:Nx] * zx[i,1:Nx] ) # regression summation, for use in likelihood
 }

 # intercept prior
 zbeta0 ~ dnorm( priorMu0 , 1/priorVar0^(2) ) # no reason to scale
 zbeta[1] ~ dnorm(zPriorMu[1], 1/zPriorVar[1]^(2))
 zbeta[2] ~ dnorm(zPriorMu[2], 1/zPriorVar[2]^(2))
 zbeta[3] ~ dnorm(zPriorMu[3], 1/zPriorVar[3]^(2))
 zbeta[4] ~ dnorm(zPriorMu[4], 1/zPriorVar[4]^(2))
 zbeta[5] ~ dnorm(zPriorMu[5], 1/zPriorVar[5]^(2))

 # prior for var in final gamma distribution, assume no information
 zVar ~ dgamma( 0.0001 , 0.0001 )

 # Transform to original scale:
```

```
 beta[1:(Nx-1)] <- ( zbeta[1:(Nx-1)] / xsd[1:(Nx-1)] ) * ysd

 beta[Nx] <- zbeta[Nx] * ysd

 beta0 <- zbeta0*ysd

 tau <- zVar*ysd


 # Compute predictions at every step of the MCMC

 for ( i in 1:5 ) {

   pred[i] <- beta0 + beta[1] * xPred[i,1] + beta[2] * xPred[i,2] +

      beta[3] * xPred[i,3] + beta[4] * xPred[i,4] + beta[5] * xPred[i,5]

 }


}

" # close quote for modelString

writeLines( modelString , con="TEMPmodel.txt" )


adaptSteps = 3000   # Number of steps to "tune" the samplers

burnInSteps = 4000  # Burn-in gives time for chains to overlap

nChains = 2

thinSteps = 12      # Reduces autocorrelation

numSavedSteps = 4000 # Save after thinning


###############################################

# RUN MCMC


# Parallel run - instead of jags.model and burn-in

startTime = proc.time()

runJagsOut <- run.jags( method="parallel" ,

            model="TEMPmodel.txt" ,

            monitor=c( "zbeta0" ,  "zbeta" , "beta0" ,  "beta" ,  "tau", "zVar", "pred"),

            data=dataList ,
                #inits=initsList ,
                n.chains=nChains ,
                adapt=adaptSteps ,
                burnin=burnInSteps ,
                sample=numSavedSteps ,
                thin=thinSteps ,
                summarise=FALSE ,
                plots=FALSE )
codaSamples = as.mcmc.list( runJagsOut )
stopTime = proc.time()
elapsedTime = stopTime - startTime
show(elapsedTime)

##################################################
# DIAGNOSIS CHECKS

# Checking:
# 1 - chains overlap
# 2 - autocorrelation low (ESS high)
# 3 - Shrink factor approaches 0
# 4 - densities overlap, (low error score also)
diagMCMC( codaSamples , parName="beta0" )
diagMCMC( codaSamples , parName="beta[1]" )
diagMCMC( codaSamples , parName="beta[2]" )
diagMCMC( codaSamples , parName="beta[3]" )
diagMCMC( codaSamples , parName="beta[4]" )
diagMCMC( codaSamples , parName="beta[5]" )
diagMCMC( codaSamples , parName="tau" )

##################################################
```

```
# SUMMARISE AND VIEW THE POSTERIORS


# Centre the posterior plots at these values
compVal <- data.frame("beta0"   = summary(codaSamples)$statistics[,"Mean"][["beta0"]],
              "beta[1]" = summary(codaSamples)$statistics[,"Mean"][["beta[1]"]],
              "beta[2]" = summary(codaSamples)$statistics[,"Mean"][["beta[2]"]],
              "beta[3]" = summary(codaSamples)$statistics[,"Mean"][["beta[3]"]],
              "beta[4]" = summary(codaSamples)$statistics[,"Mean"][["beta[4]"]],
              "beta[5]" = summary(codaSamples)$statistics[,"Mean"][["beta[5]"]],
              "tau"     = summary(codaSamples)$statistics[,"Mean"][["tau"]],
              "pred[1]" = summary(codaSamples)$statistics[,"Mean"][["pred[1]"]],
              "pred[2]" = summary(codaSamples)$statistics[,"Mean"][["pred[2]"]],
              "pred[3]" = summary(codaSamples)$statistics[,"Mean"][["pred[3]"]],
              "pred[4]" = summary(codaSamples)$statistics[,"Mean"][["pred[4]"]],
              "pred[5]" = summary(codaSamples)$statistics[,"Mean"][["pred[5]"]],
              check.names=FALSE)


summaryInfo <- smryMCMC(codaSamples=codaSamples,
               compVal=compVal)
print(summaryInfo)


plotMCMC_HD(codaSamples=codaSamples,
       data=myData,
       xName=c("Area","Bedrooms","Bathrooms","CarParks","PropertyType") ,
       yName="SalePrice.100K.",
       compVal = compVal)
```