Rain Project

Luca

August 31, 2020

1. Load Data

This dataset contains daily weather observations from numerous Australian weather stations.

The target variable RainTomorrow means: Did it rain the next day? Yes or No.

```r
data <- read.table("C:\\Users\\Luca\\Desktop\\SDS 2 Project\\Data\\weatherAUS.csv",
                   sep = ",", header = T, fill = T)


head(data)
```

```
##         Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine WindGustDir
## 1 2008-12-01   Albury    13.4    22.9      0.6          NA       NA           W
## 2 2008-12-02   Albury     7.4    25.1      0.0          NA       NA         WNW
## 3 2008-12-03   Albury    12.9    25.7      0.0          NA       NA         WSW
## 4 2008-12-04   Albury     9.2    28.0      0.0          NA       NA          NE
## 5 2008-12-05   Albury    17.5    32.3      1.0          NA       NA           W
## 6 2008-12-06   Albury    14.6    29.7      0.2          NA       NA         WNW
##   WindGustSpeed WindDir9am WindDir3pm WindSpeed9am WindSpeed3pm Humidity9am
## 1            44          W        WNW           20           24          71
## 2            44        NNW        WSW            4           22          44
## 3            46          W        WSW           19           26          38
## 4            24         SE          E           11            9          45
## 5            41        ENE         NW            7           20          82
## 6            56          W          W           19           24          55
##   Humidity3pm Pressure9am Pressure3pm Cloud9am Cloud3pm Temp9am Temp3pm
## 1          22      1007.7      1007.1        8       NA    16.9    21.8
## 2          25      1010.6      1007.8       NA       NA    17.2    24.3
## 3          30      1007.6      1008.7       NA        2    21.0    23.2
## 4          16      1017.6      1012.8       NA       NA    18.1    26.5
## 5          33      1010.8      1006.0        7        8    17.8    29.7
## 6          23      1009.2      1005.4       NA       NA    20.6    28.9
##   RainToday RISK_MM RainTomorrow
## 1        No     0.0           No
## 2        No     0.0           No
## 3        No     0.0           No
## 4        No     1.0           No
## 5        No     0.2           No
## 6        No     0.0           No
```

```r
dim(data)
```

```
## [1] 142193     24
```

```r
# Delete NaN values
clean_data = na.omit(data)
dim(clean_data)
```

```
## [1] 56420    24
```

```r
# Analyze the variables involved
str(clean_data)
```

```
## 'data.frame':    56420 obs. of  24 variables:
##  $ Date         : chr  "2009-01-01" "2009-01-02" "2009-01-04" "2009-01-05" ...
##  $ Location     : chr  "Cobar" "Cobar" "Cobar" "Cobar" ...
##  $ MinTemp      : num  17.9 18.4 19.4 21.9 24.2 27.1 23.3 16.1 19 19.7 ...
##  $ MaxTemp      : num  35.2 28.9 37.6 38.4 41 36.1 34 34.2 35.5 35.5 ...
##  $ Rainfall     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Evaporation  : num  12 14.8 10.8 11.4 11.2 13 9.8 14.6 12 11 ...
##  $ Sunshine     : num  12.3 13 10.6 12.2 8.4 0 12.6 13.2 12.3 12.7 ...
##  $ WindGustDir  : chr  "SSW" "S" "NNE" "WNW" ...
##  $ WindGustSpeed: int  48 37 46 31 35 43 41 37 48 41 ...
##  $ WindDir9am   : chr  "ENE" "SSE" "NNE" "WNW" ...
##  $ WindDir3pm   : chr  "SW" "SSE" "NNW" "WSW" ...
##  $ WindSpeed9am : int  6 19 30 6 17 7 17 15 30 15 ...
##  $ WindSpeed3pm : int  20 19 15 6 13 20 19 6 9 17 ...
##  $ Humidity9am  : int  20 30 42 37 19 26 33 25 46 61 ...
##  $ Humidity3pm  : int  13 8 22 22 15 19 15 9 28 14 ...
##  $ Pressure9am  : num  1006 1013 1012 1013 1011 ...
##  $ Pressure3pm  : num  1004 1012 1009 1009 1007 ...
##  $ Cloud9am     : int  2 1 1 1 1 8 3 1 1 1 ...
##  $ Cloud3pm     : int  5 1 6 5 6 8 1 1 5 5 ...
##  $ Temp9am      : num  26.6 20.3 28.7 29.1 33.6 30.7 25 20.7 23.4 24 ...
##  $ Temp3pm      : num  33.4 27 34.9 35.6 37.6 34.3 31.5 32.8 33.3 33.6 ...
##  $ RainToday    : chr  "No" "No" "No" "No" ...
##  $ RISK_MM      : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ RainTomorrow : chr  "No" "No" "No" "No" ...
##  - attr(*, "na.action")= 'omit' Named int [1:85773] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "names")= chr [1:85773] "1" "2" "3" "4" ...
```

2. Feauture Selection & Data Analysis

As we can read on the dataset's documentation we should remove the feauture RISK_MM. This variable shows the amount of next day rain in mm (millimetre) and not excluding it will leak the answers to your model and reduce its predictability. It's highly correlated to the target value, and thus we'll drop it. Looking at variables we can see that they are made up by different data types, we'll drop some of them like **Location** and others should be converted from string to boolean when we find "Yes" and "No", like **RainToday** and **RainTomorrow**.

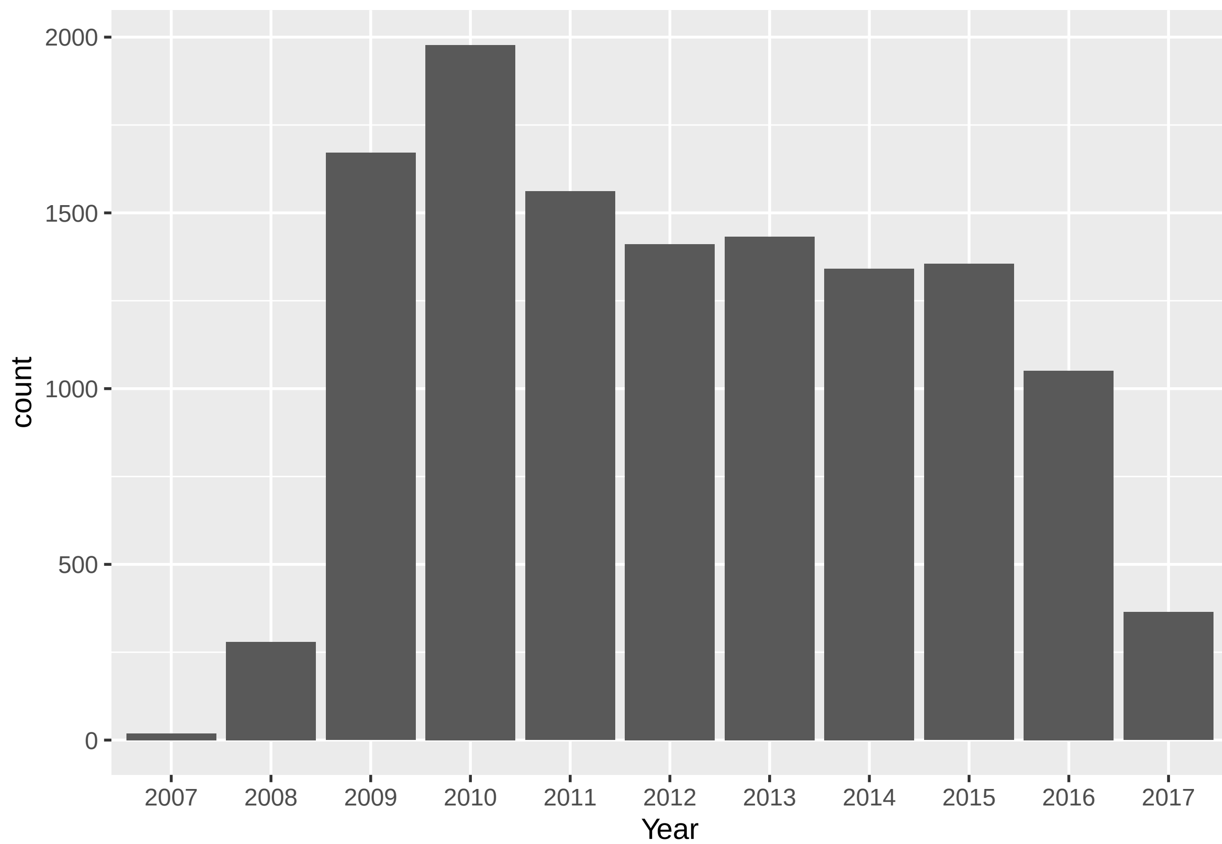
```r
clean_data$RISK_MM <- NULL

# We would like to transform two variables from character to Boolean
# Yes --> 1 and No --> 0

clean_data <- clean_data %>%
  mutate(RainToday = ifelse(RainToday == "No",0,1))

clean_data <- clean_data %>%
  mutate(RainTomorrow = ifelse(RainTomorrow == "No",0,1))

str(clean_data)
```

```
## 'data.frame':    56420 obs. of  23 variables:
##  $ Date         : chr  "2009-01-01" "2009-01-02" "2009-01-04" "2009-01-05" ...
##  $ Location     : chr  "Cobar" "Cobar" "Cobar" "Cobar" ...
##  $ MinTemp      : num  17.9 18.4 19.4 21.9 24.2 27.1 23.3 16.1 19 19.7 ...
##  $ MaxTemp      : num  35.2 28.9 37.6 38.4 41 36.1 34 34.2 35.5 35.5 ...
##  $ Rainfall     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Evaporation  : num  12 14.8 10.8 11.4 11.2 13 9.8 14.6 12 11 ...
##  $ Sunshine     : num  12.3 13 10.6 12.2 8.4 0 12.6 13.2 12.3 12.7 ...
##  $ WindGustDir  : chr  "SSW" "S" "NNE" "WNW" ...
##  $ WindGustSpeed: int  48 37 46 31 35 43 41 37 48 41 ...
##  $ WindDir9am   : chr  "ENE" "SSE" "NNE" "WNW" ...
##  $ WindDir3pm   : chr  "SW" "SSE" "NNW" "WSW" ...
##  $ WindSpeed9am : int  6 19 30 6 17 7 17 15 30 15 ...
##  $ WindSpeed3pm : int  20 19 15 6 13 20 19 6 9 17 ...
##  $ Humidity9am  : int  20 30 42 37 19 26 33 25 46 61 ...
##  $ Humidity3pm  : int  13 8 22 22 15 19 15 9 28 14 ...
##  $ Pressure9am  : num  1006 1013 1012 1013 1011 ...
##  $ Pressure3pm  : num  1004 1012 1009 1009 1007 ...
##  $ Cloud9am     : int  2 1 1 1 1 8 3 1 1 1 ...
##  $ Cloud3pm     : int  5 1 6 5 6 8 1 1 5 5 ...
##  $ Temp9am      : num  26.6 20.3 28.7 29.1 33.6 30.7 25 20.7 23.4 24 ...
##  $ Temp3pm      : num  33.4 27 34.9 35.6 37.6 34.3 31.5 32.8 33.3 33.6 ...
##  $ RainToday    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ RainTomorrow : num  0 0 0 0 0 0 0 0 0 0 ...
##  - attr(*, "na.action")= 'omit' Named int [1:85773] 1 2 3 4 5 6 7 8 9 10 ...
##   ..- attr(*, "names")= chr [1:85773] "1" "2" "3" "4" ...
```

What is the year with the most rainy days?

Before we delete the feauture **Date** we should use it to perform some Exploratory Data Analysis.

```r
clean_data$Date <- as.Date(clean_data$Date)
# Subset of the data formed only by rainy days
newdata <- subset(clean_data, clean_data$RainToday == 1)
ggplot(newdata, aes(format(newdata$Date, "%Y"))) +
  geom_bar(stat = "count") +
  labs(x = "Year")
```

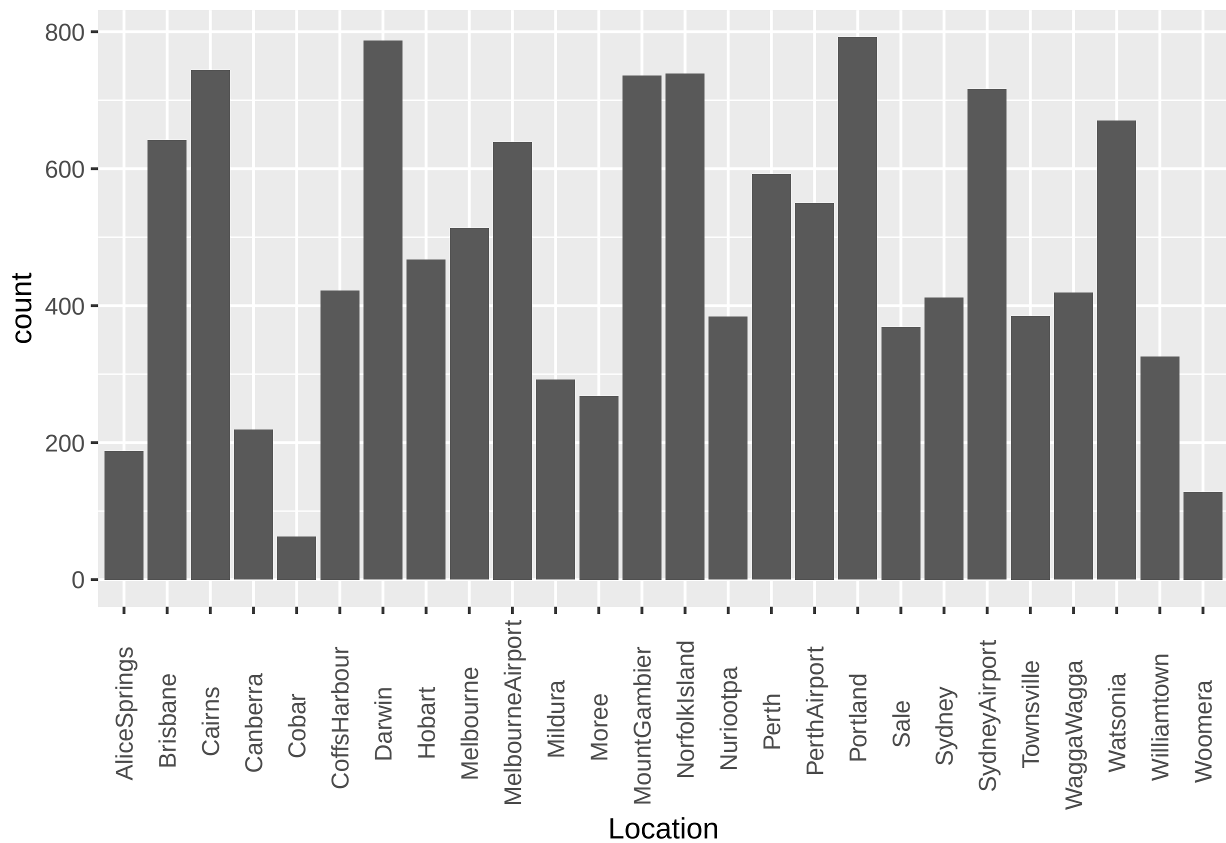



```
clean_data$Date <- NULL
```

In 2010, Australia experienced its third-wettest year since national rainfall records began in 1900

Which is the location where it rained the most days?

```
ggplot(newdata, aes(format(newdata$Location))) +
  geom_bar(stat = "count") +
  labs(x = "Location")+
  theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1))
```

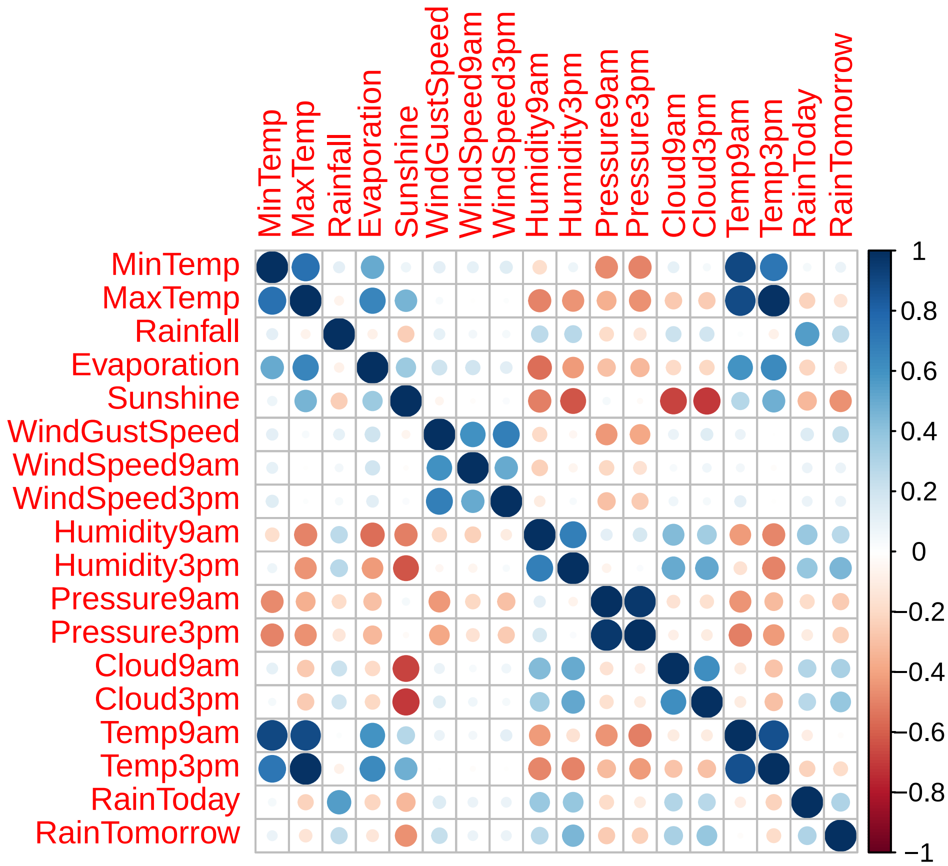



```
clean_data$Location <- NULL

# Other variables that can't be converted to numeric
clean_data$WindGustDir <- NULL
clean_data$WindDir3pm <- NULL
clean_data$WindDir9am <- NULL
```

Correlation plot

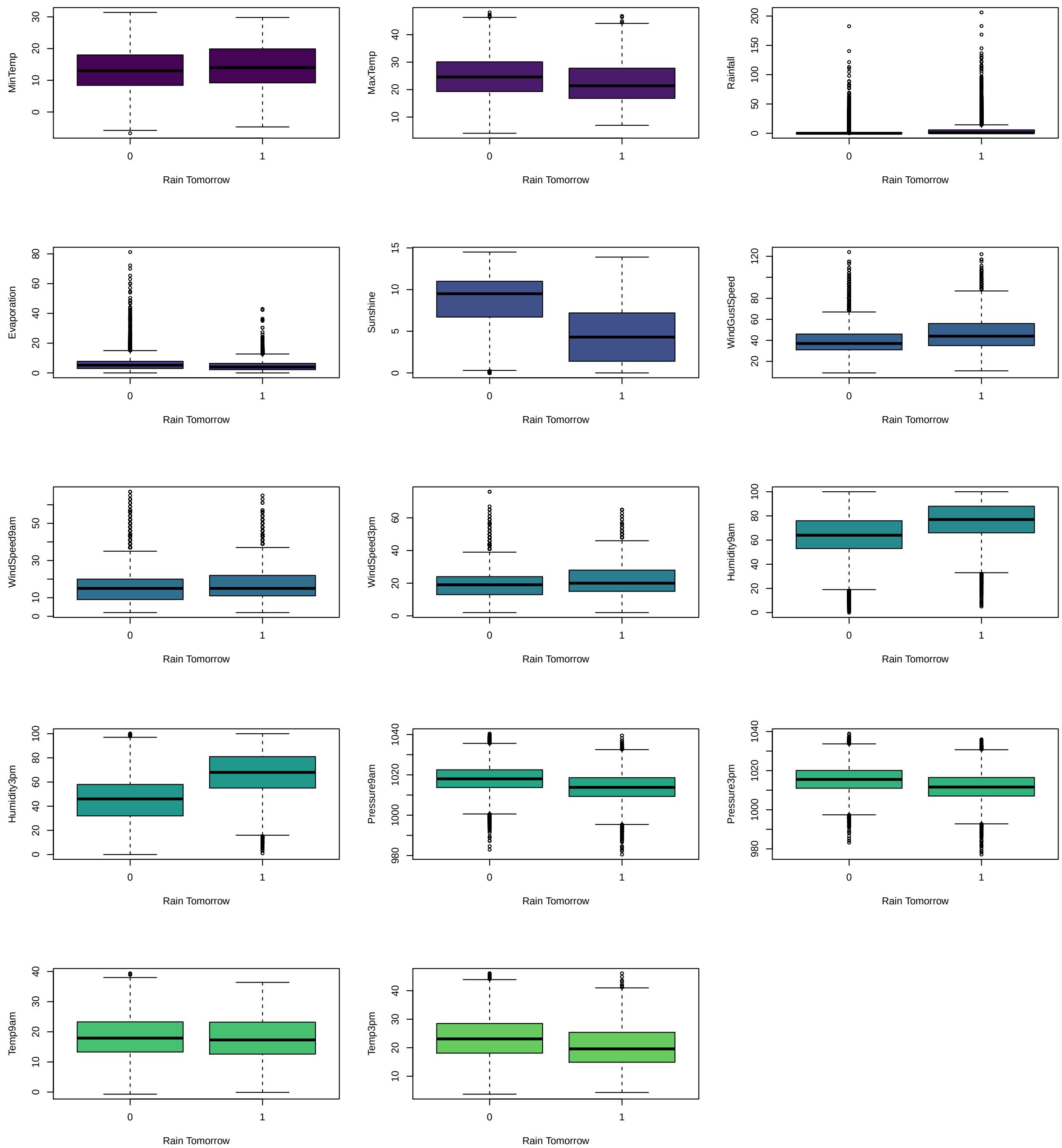
This correlation plot shows us that some variables are higly correlated to each other. This is not great news, since they'll compete to explain the Y variable, we'll later delete some of these after we see some more proof.

```
knitr::opts_chunk$set(fig.width=12, fig.height=8)
par(mfrow = c(1,1))
correlations <- cor(clean_data)
corrplot(correlations, method="circle")
```



Plots of feautures vs response variable

```
attach(clean_data)
#knitr::opts_chunk$set(fig.width=0.5, fig.height=0.5)
colors = viridis(18, alpha = 0.99)

columns <- colnames(clean_data)

par(mfrow = c(3,3))

j = 1
for(i in columns){

  if(i=="Cloud9am" || i=="Cloud3pm" || i=="RainToday" || i=="RainTomorrow") next


  boxplot(clean_data[[i]] ~ RainTomorrow ,
       data = clean_data, col = colors[j], xlab = "Rain Tomorrow",          ylab = i)
  j <- j + 1

}
```


MinTemp
MaxTemp
Rainfall
Evaporation
Sunshine
WindGustSpeed
WindSpeed9am
WindSpeed3pm
Humidity9am
Humidity3pm
Pressure9am
Pressure3pm
Temp9am
Temp3pm
Rain Tomorrow
0
1

3. To do list:

We have a couple of steps to follow before creating the model:

- 3.1 Normalize X's (subtracting the mean and dividing by the SD)
- 3.2 Skimming through variables
- 3.3 Description of the diagnostics we'll perform

3.1 Normalization

```
# Standardize variables
index <- sample(1:nrow(clean_data), 10000) # We are going to take 20.000 samples out of the initial dat
new_data <- clean_data[index,]

# We are going to scale continous values by subtracting the mean and dividing by the SD
X = scale(new_data[,-18],center = TRUE, scale = TRUE)
```

```
y = new_data$RainTomorrow # Target variable
X = na.omit(X) # Delete all Nan values (double checking)
```

y_i is a Bernoulli outcome [0,1], we are going to use a logarithmic scale as a link function that relates the linear form of the restricted parameter to a [0,1] interval. We need to find the probability of success, by doing this we need to apply Bayesian Methods.

Main Ingredients:

$$Likelihood = (y_i|\phi) \sim Bern(\phi_i)$$

Since ϕ is the probability of success (Tomorrow it will rain), then:

$$E(\phi_i) = p$$

$$E(yi) = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

We need to use the link function that relates the linear form of the restricted parameter and allow us to have our $\phi \epsilon$ [0, 1]. Thus:

$$logit(\phi_i) = log\frac{\phi}{(1-\phi)} = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

If we use some algebra we can rewrite these equations as:

$$logit(\phi_i) = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n \Rightarrow \phi_i = \frac{e^{\beta_0 + \beta_1 x_1 + ... + \beta_n x_n}}{1+e^{\beta_0 + \beta_1 x_1 + ... + \beta_n x_n}}$$

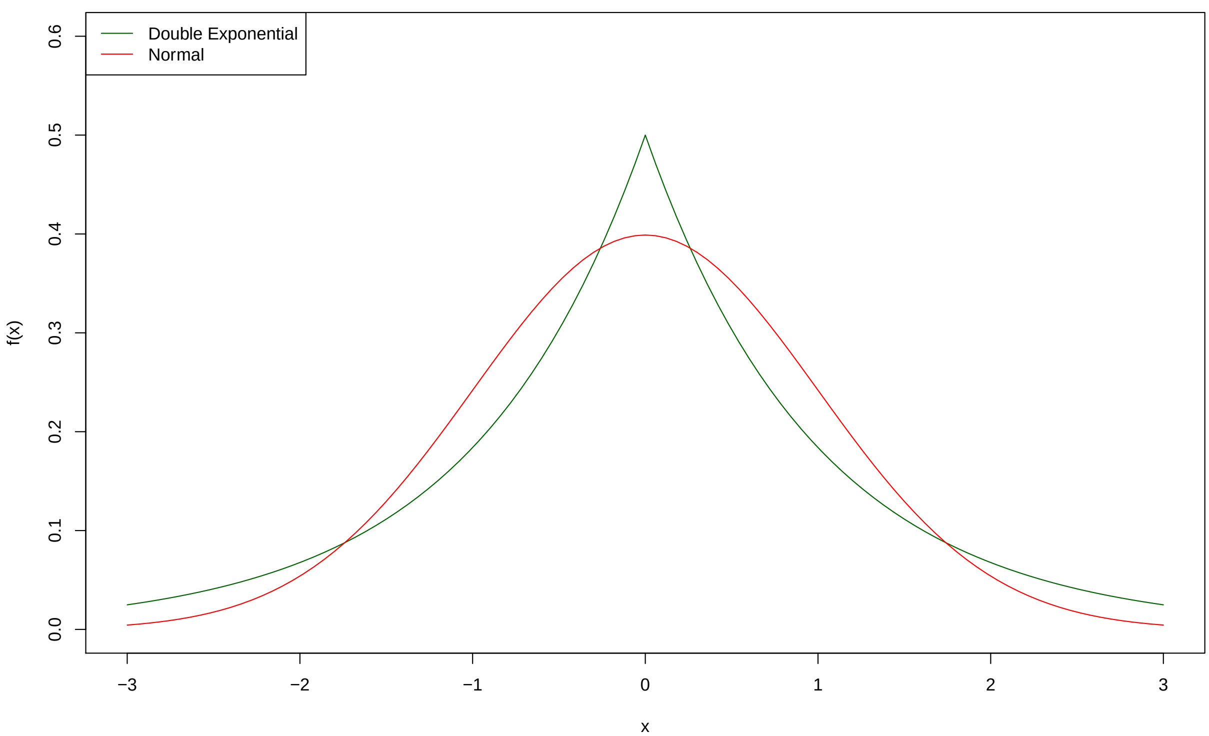
At the end we can rewrite this as:

$$\phi_i = \frac{1}{1+e^{-(\beta_0 + \beta_1 x_1 + ... + \beta_n x_n)}}$$

Visualization of different distributions

Since we are going to use two different models, with two different prior distributions, I would like to show two generic plots of these distributions.

```r
# Visualize double exponential function
f <- function(x){ddexp(x)}
g <- function(x){dnorm(x)}
curve(f(x),xlim = c(-3,3),ylim=c(0,0.6),col = 'dark green')
curve(g(x),add = T,xlim = c(-3,3),ylim=c(0,0.6),col = 'red')
# Add a legend
legend("topleft", legend = c("Double Exponential", "Normal"),
  col = c('dark green', 'red'), lty=1)
```



The plot points out the main difference of a Normal distribution and a double exponential one. Both are higly concentrated on 0, but the double exponential one has heavier tails and is more concentrated on 0.

3.2 Feature selection

Now we can create the model -> GLM (Generalized linear model) and we are going to pick family = LOGIT.

```r
glm.fit <- glm(y ~ X[,1] + X[,2] + X[,3] + X[,4] + X[,5]+
                 X[,6] + X[,7]+X[,8]+X[,9]+X[,10]+X[,11]+
                 X[,12]+X[,13]+X[,14]+X[,15]+X[,16], family =
                 'binomial'(link="logit"), maxit = 50)

summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ X[, 1] + X[, 2] + X[, 3] + X[, 4] + X[, 5] +
```

```
##     X[, 6] + X[, 7] + X[, 8] + X[, 9] + X[, 10] + X[, 11] + X[,
##     12] + X[, 13] + X[, 14] + X[, 15] + X[, 16], family = binomial(link = "logit"),
##     maxit = 50)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2397  -0.5030  -0.2805  -0.1164   3.2864
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.96000    0.04065 -48.217  < 2e-16 ***
## X[, 1]      -0.36473    0.11577  -3.150  0.00163 **
## X[, 2]      -0.01564    0.19167  -0.082  0.93496
## X[, 3]       0.22205    0.03554   6.248 4.16e-10 ***
## X[, 4]      -0.06437    0.05121  -1.257  0.20878
## X[, 5]      -0.54378    0.05514  -9.861  < 2e-16 ***
## X[, 6]       0.89307    0.05190  17.206  < 2e-16 ***
## X[, 7]      -0.12119    0.04179  -2.900  0.00373 **
## X[, 8]      -0.28649    0.04584  -6.250 4.11e-10 ***
## X[, 9]       0.16341    0.06992   2.337  0.01943 *
## X[, 10]      1.15092    0.08156  14.111  < 2e-16 ***
## X[, 11]      0.82821    0.13090   6.327 2.50e-10 ***
## X[, 12]     -1.23057    0.13122  -9.378  < 2e-16 ***
## X[, 13]     -0.04432    0.04963  -0.893  0.37187
## X[, 14]      0.24283    0.05152   4.714 2.43e-06 ***
## X[, 15]      0.46728    0.17557   2.661  0.00778 **
## X[, 16]     -0.02479    0.21350  -0.116  0.90756
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 10523.0  on 9999  degrees of freedom
## Residual deviance:  6600.7  on 9983  degrees of freedom
## AIC: 6634.7
##
## Number of Fisher Scoring iterations: 6
```

This is a first skimming of some variables, from now on we'll only keep the ones that influece the response variable.

3.3 Diagnostics we are going to use

- **Trace & Density plot:** The target of this plot is to show random scatter around the mean value, and our model results suggest that the chains mixed well and the traceplot looked satisfactory. One reason for running multiple chains is that any individual chain might converge toward one target, while another chain might converge elsewhere and this would still be a problem. Also, you might see healthy chains getting stuck over the course of the series, which might suggest more model tweaking or a change in the sampler settings is warranted. The density plots are used as a graphical assessment for the coefficients. The goal is to have normally distributed random variables with posterior mean different from 0, otherwise this signifies that our data didn't give us additional information than the one we had with the prior double exponential one (centered in 0).

- **Autocorrelation:** This plots shows us for each lag the correlation between the current one and the previous one. The goal is to have a plot that drops as soon as possible, the sooner it is the less iterations we need for our MCMC. We could double check this with the effective sample size, it's the number of independet observations our sample is equivalent to. The greater the correlation between observations, the smallest the effective sample size will be.
- **Gelman Rubin:** It uses different starting values that are overdispersed relative to the posterior distribution. Convergence is diagnosed when the chains have "forgotten" their initial values, and the output from all chains is indistringuishable, in other words all chains should have the dame distribution. It is based on comparison of within chain and between chain variance and what we want is less between chain rather than within, this means we got to a convergence point. Analytically:

We have M chains of length N and a parameter ϕ. For each chain we have $\{\phi_{mt}\}_{t=1}^{N}$ where:

$$\hat{\phi_m} = Posterior\ Mean$$

$$\hat{\sigma^2} = PosteriorVariance$$

At this point we can compute the overall Posterior Mean $= \hat{\phi} = \frac{1}{M}\sum_{n=1}^{M}\hat{\phi_m}$

We need all of these ingredients to compute the **Between** Variance and **Within** Variance of each chain in order to obtain the **Pooled Variance**:

$$\text{Between variance} = B = \frac{N}{M-1}\sum_{m=1}^{M}(\hat{\phi_m} - \hat{\phi})^2$$

$$\text{Within variance} = W = \frac{1}{M}\sum_{m=1}^{M}\hat{\sigma_m^2}$$

$$\text{Pooled Variance} = \hat{V} = \frac{N-1}{N}W + \frac{M+1}{MN}B$$

This was the second-last step performed by this algorithm, now the only thing left is to compute the ration between $\hat{V}$ and W. If this ration is close to 1 (not more than 1.1), then we've obtained convergence.

- **Heidelberg-Welch:** It's based on the assumption that we have got a **weakly stationary process** when the chain has reached convergence. This means that:

$$E[x_j] \text{ is constant throughout time}$$

$$Cov(\phi^j, \phi^{j+s}) \text{ does not depend on j}$$

This diagnostic not only tells us whether the MCMC converged, but also if we had run the chain enough times. The first part tests a null hypothesis that the sampled values of the chain come from a stationary distribution. If the hypothesis is rejected than we discard the first 10%, then 20%, until either the null hypothesis is accepted or 50% of the chain has been discarded. If the stationary test is passed, the number of iterations to keep and the number to discard are reported.

Then we pass to the second part, half width test. It will tell us if we can estimate the mean with some level of accuracy. We compute the ratio of the margin of error (Halfwidth/Mean)and compare it to the estimated mean. If the ration is less than epsilon (usually 0.1) than the test is passed, otherwise we should extend the chain because we aren't able to estimate the mean.

- **Raftery:** This test has to approve a couple of conditions. We would like to compute a posterior quantile **q** with some tollerance **r** (+ or -). We then pick a probability **s** which is the probability of being within the interval **(q-r, q+r)**. This diagnostic test estimates the number N of iterations and the number of burn-in iterations that are necessary in order to satisfy these two conditions.

$$z = \frac{\bar{\phi_a} - \bar{\phi_b}}{\sqrt{Var(\phi_a) - Var(\phi_b)}}$$

where **a = initial interval and b = late interval** and z should fall within two standard deviation of zero.

- **Geweke:** Compare the estimate of the mean of the first part of the chain with the last part of the chain. By default the first part is 10% and the latter part is 50%. If they come from the same stationary distribution then the mean should be equal and the Geweke's statistics has an asymptotically standard normal distribution. This will produce a test statistics, if this passes we should get a value between -2 and +2 (this means that the chain has converged to its stationary distribution)

4 Model n.1 using Normal prior (RIDGE)

This model shows us the first part that corresponds to the likelihood function and we'll use a Bernoulli distribution to model y[i]. We won't model directly the prob of success = p[i], but the logit of that function, get's the linear part of the function. We'll use a non informative prior that is a Normal with mean = 0 and SD = 0.00010.

We are going to use three different chains to check wether they all converge to a stable solution.

```
# Let's write down the model
inits_0 = list("int" = 0.2, 'lambda' = 0.2, 'b' = rep(0.1,10))
inits_1 = list("int" = 0.15, 'lambda' = 0.15, 'b' = rep(0.05,10))
inits_2 = list("int" = 0.25, 'lambda' = 0.25, 'b' = rep(0.2,10))
inits_total1 = list(inits_0,inits_1,inits_2)

mod1_string = "model{

  for (i in 1:length(y)){
    y[i] ~ dbern(p[i])

    logit(p[i]) = int + b[1]*Evaporation[i] + b[2]*Sunshine[i] + b[3]*WindGustSpeed[i] + b[4]*WindSpeed
  }

  int ~ dnorm(0.0,1.0E-6)

  for (j in 1:10){
    b[j] ~ dnorm(0.0,lambda) # prior,has variance 1
  }
  lambda ~ dgamma(0.1,0.1)

}"

set.seed(123)
data_jags = list(y = new_data$RainTomorrow,
                 Evaporation = X[,4],
                 Sunshine = X[,5],
                 WindGustSpeed = X[,6],
```

```r
                  WindSpeed9am = X[,7],
                  Humidity9am = X[,9],
                  Humidity3pm = X[,10],
                  Pressure9am = X[,11],
                  Pressure3pm = X[,12],
                  Cloud9am = X[,13],
                  Temp3pm = X[,16])

params = c('int','b','lambda')
mod1 = jags(data = data_jags, inits = inits_total1,
            parameters.to.save = params, model.file = textConnection(mod1_string),n.chains = 3,
            n.iter = 9000)
```

```
## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 10000
##    Unobserved stochastic nodes: 12
##    Total graph size: 131830
##
## Initializing model
```

```r
# The function `mcmc' is used to create a Markov Chain Monte Carlo object.
mod1_sim = as.mcmc(mod1)
options(scipen=999)
```

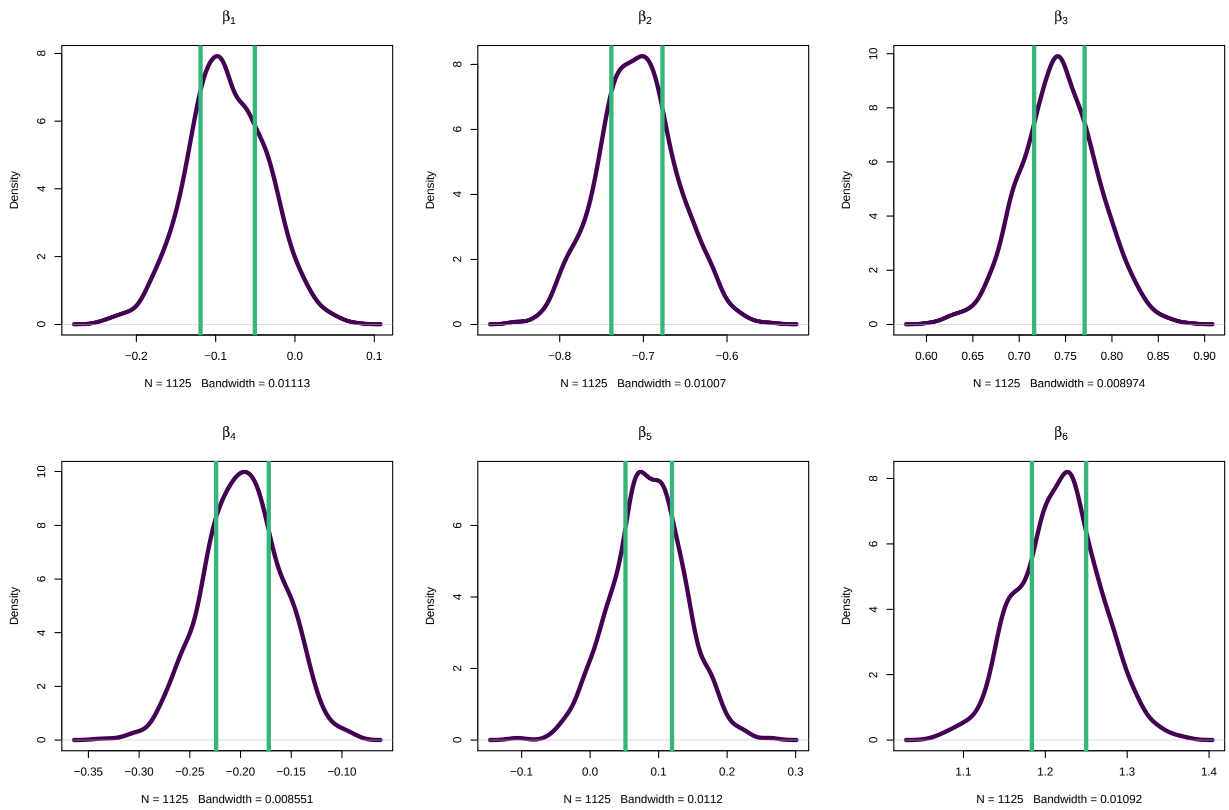
```r
mod1$BUGSoutput$summary
```

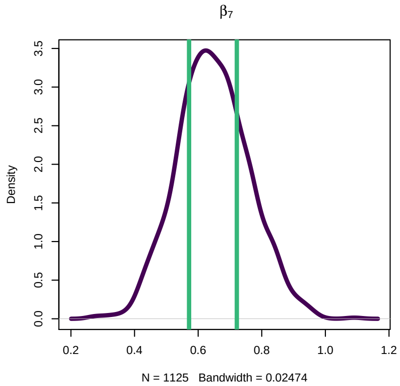
```
##                   mean         sd         2.5%         25%         50%
## b[1]       -0.08684714 0.04979898   -0.185789586   -0.11973740   -0.08814478
## b[2]       -0.70694827 0.04781761   -0.798267818   -0.73924038   -0.70653103
## b[3]        0.74258794 0.04316348    0.658728663    0.71301332    0.74287903
## b[4]       -0.19773791 0.03892570   -0.274449582   -0.22399678   -0.19751265
## b[5]        0.08504632 0.05202892   -0.014541815    0.05028723    0.08472734
## b[6]        1.21428906 0.05122698    1.115553455    1.17994680    1.21374170
## b[7]        0.64378723 0.11553332    0.423499747    0.56557018    0.64180916
## b[8]       -1.09195116 0.11796132   -1.322323353   -1.17186705   -1.09058288
## b[9]       -0.03971509 0.04814217   -0.134741083   -0.07212689   -0.03904700
## b[10]       0.08877193 0.04911004   -0.007889403    0.05521124    0.08890086
## deviance 6738.06544402 4.85941981 6730.741211559 6734.46591205 6737.32611187
## int        -1.97309984 0.03902546   -2.048155343   -2.00001532   -1.97310727
## lambda      2.31416522 1.05396497    0.719449264    1.54336267    2.13891142
##                    75%         97.5%     Rhat n.eff
## b[1]       -0.052902067    0.01130623 1.001486   2100
## b[2]       -0.675132932   -0.61358603 1.000644   3400
## b[3]        0.771161388    0.82775441 1.000836   3400
## b[4]       -0.171366852   -0.12343379 1.000746   3400
## b[5]        0.119247730    0.18781603 1.002154   1300
## b[6]        1.249079199    1.31303150 1.001826   1600
```

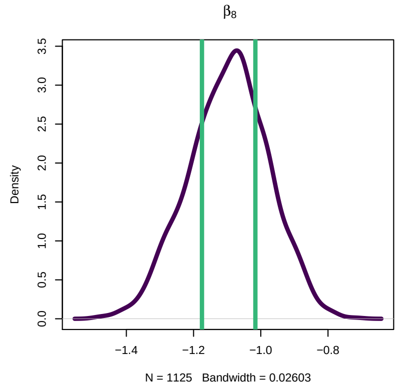
```
## b[7]         0.721723882    0.86526996 1.001577  2000
## b[8]        -1.012852538   -0.86351666 1.001166  3300
## b[9]        -0.007527533    0.05198744 1.000637  3400
## b[10]        0.123047615    0.18245500 1.000794  3400
## deviance 6740.920063738 6749.39561009 1.001393  2400
## int         -1.946346543   -1.89756022 1.003307   730
## lambda       2.924456084    4.80853005 1.000652  3400
```

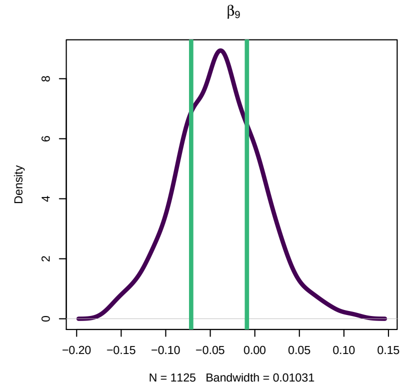
Feauture selection with Confidence Intervals

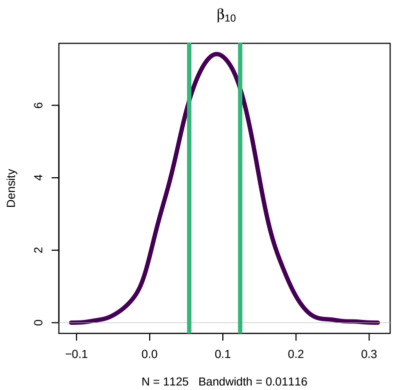
One suggested method for doing that exploits the use of the credible intervals of the posterior of the β_i: if, for a fixed level α, the interval contains 0, then the coefficients should be excluded. Generally, the author of the paper noted that the usual sets of α are too large for the variable selection and would exclude too many β_i. It is instead suggested an α of 0.5.









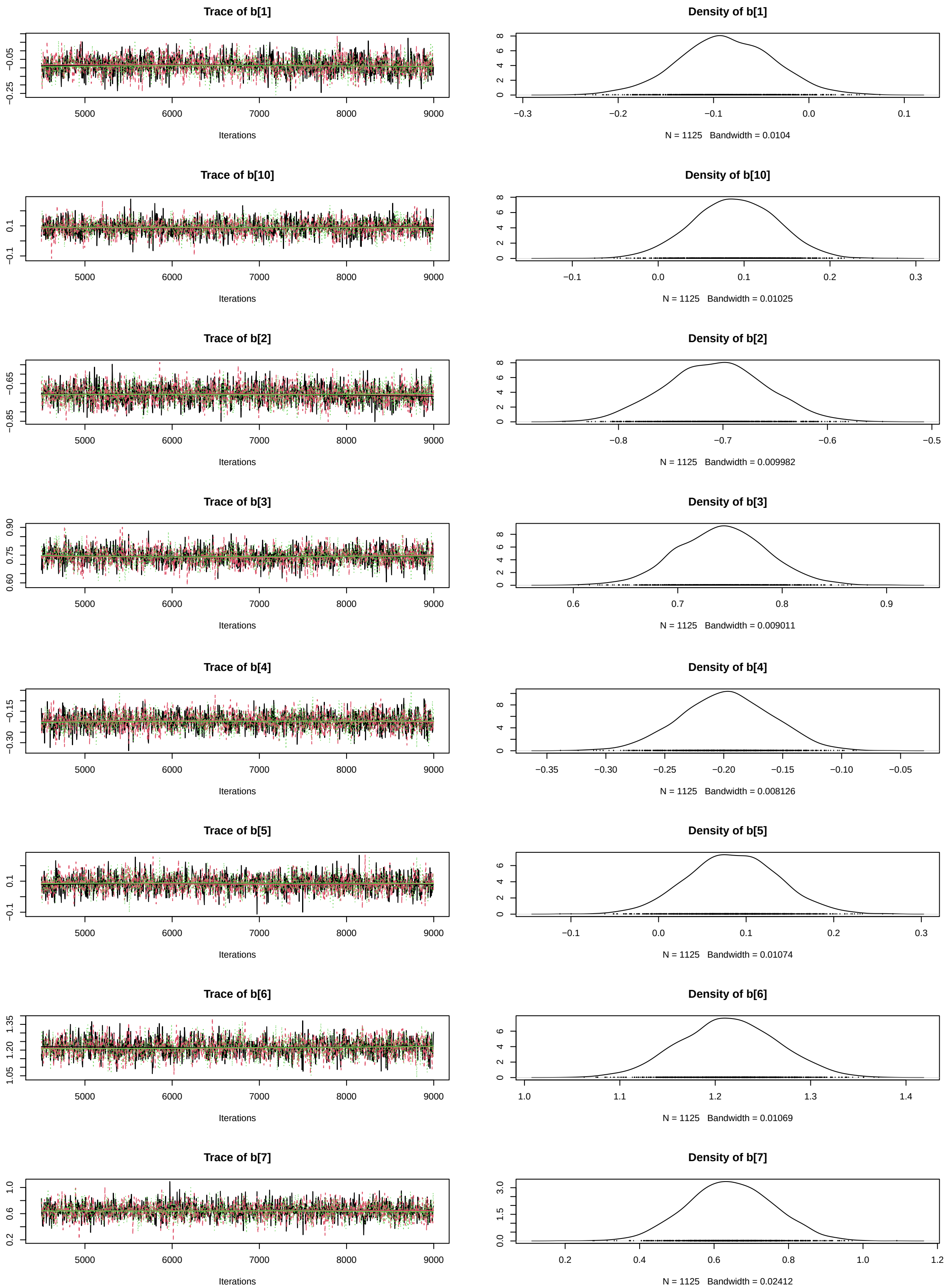


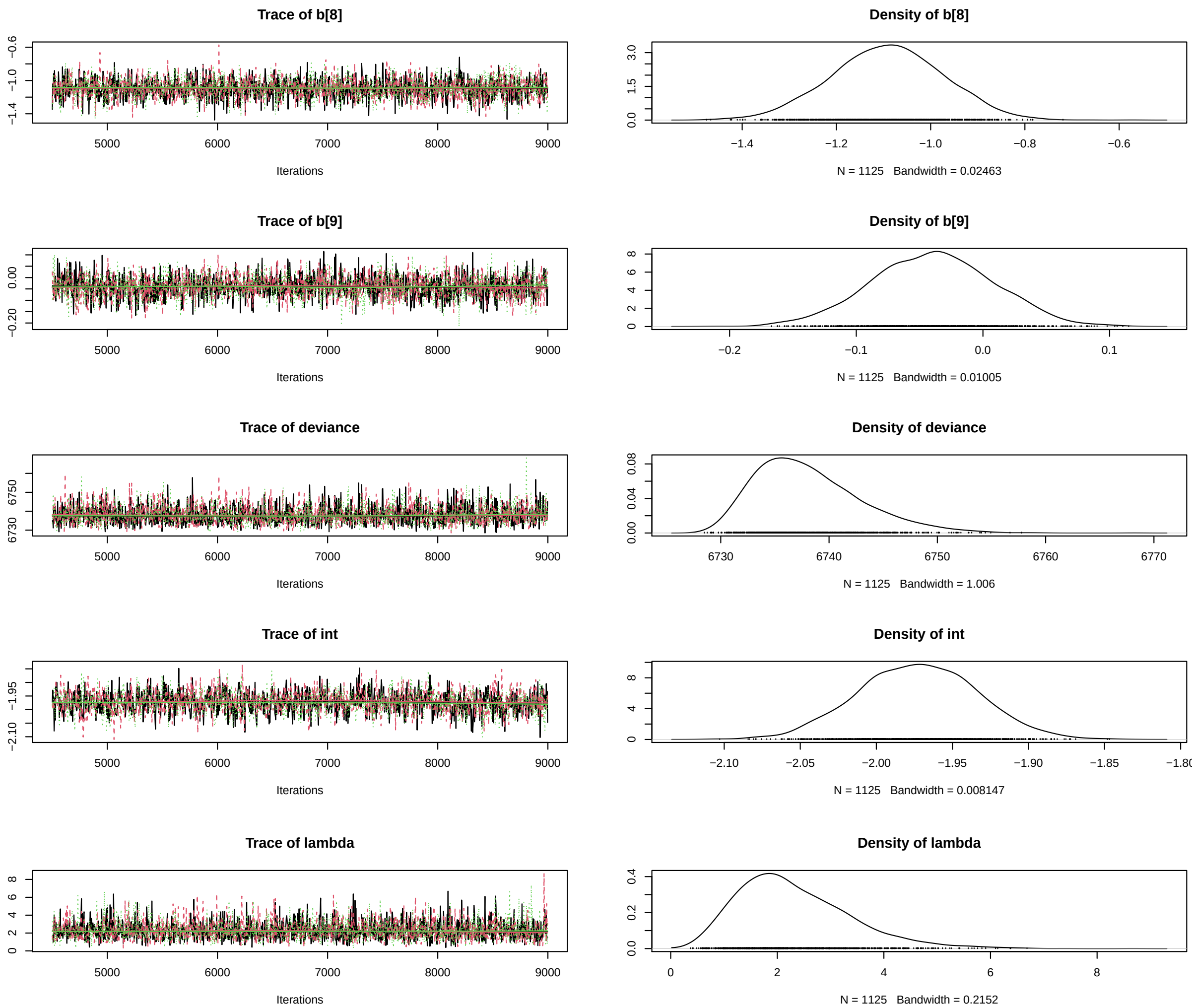
Trace & Density plots

- Humidity3pm (Beta 7)
- Pressure9am (Beta 8)
- Pressure3pm (Beta 9)
- Cloud9am (Beta 10)
- Temp3pm (Beta 11)

All of the variables listed above show some autocorrelation, as we can see in the traceplot and the density plot as well. They all have mean close to 0, this means that they won't help us predicting Y_i observations. For the second model I decided to drop these variables and keep the most significant ones.

```r
plot.new()
par(mfrow = c(3,4))
plot(mod1_sim)
```


Trace of b[1]
Iterations
Density of b[1]
N = 1125 Bandwidth = 0.0104
Trace of b[10]
Iterations
Density of b[10]
N = 1125 Bandwidth = 0.01025
Trace of b[2]
Iterations
Density of b[2]
N = 1125 Bandwidth = 0.009982
Trace of b[3]
Iterations
Density of b[3]
N = 1125 Bandwidth = 0.009011
Trace of b[4]
Iterations
Density of b[4]
N = 1125 Bandwidth = 0.008126
Trace of b[5]
Iterations
Density of b[5]
N = 1125 Bandwidth = 0.01074
Trace of b[6]
Iterations
Density of b[6]
N = 1125 Bandwidth = 0.01069
Trace of b[7]
Iterations
Density of b[7]
N = 1125 Bandwidth = 0.02412


Trace of b[8]
Iterations
Density of b[8]
N = 1125 Bandwidth = 0.02463
Trace of b[9]
Iterations
Density of b[9]
N = 1125 Bandwidth = 0.01005
Trace of deviance
Iterations
Density of deviance
N = 1125 Bandwidth = 1.006
Trace of int
Iterations
Density of int
N = 1125 Bandwidth = 0.008147
Trace of lambda
Iterations
Density of lambda
N = 1125 Bandwidth = 0.2152

Diagnostics

```
superdiag(mod1_sim, burnin = 100)
```

```
## Number of chains = 3
## Number of iterations = 1125 per chain before discarding the burn-in period
## The burn-in period = 100 per chain
## Sample size in total = 3075
##
## ********** The Geweke diagnostic: **********
## Z-scores:
##                        chain1    chain 2    chain 3
## b[1]                1.4712125 -1.3964072 -2.2992894
## b[10]              -1.8082732  0.7193819  7.4783143
## b[2]                0.8933134  0.4189543 -0.7095667
## b[3]                1.1635281 -1.8387106  1.5640419
## b[4]               -0.5451937  1.1752144 -3.2571393
## b[5]                0.4895504 -1.3019170 -0.7274969
## b[6]                1.3380850  0.9073274  1.2020849
## b[7]               -1.1238826  0.1106256 -0.6090731
## b[8]                0.2211194  0.1481677  0.7075501
## b[9]               -1.0484357 -0.6432881 -1.7045799
## deviance           -0.4081290  0.4388882 -1.0596384
## int                -0.1078269  0.3086408  2.1877945
## lambda              0.4357866 -0.2367065  0.9468483
## Window From Start   0.1000000  0.6511900  0.9471900
## Window From Stop    0.5000000  0.0158400  0.0047300
##
## ********** The Gelman-Rubin diagnostic: **********
## Potential scale reduction factors:
##
##          Point est. Upper C.I.
## b[1]          1.001      1.002
## b[10]         1.001      1.002
## b[2]          0.999      1.000
## b[3]          0.999      0.999
## b[4]          1.001      1.002
## b[5]          1.003      1.013
## b[6]          1.001      1.001
## b[7]          1.003      1.011
## b[8]          1.000      1.004
## b[9]          1.000      1.000
## deviance      1.003      1.006
## int           1.003      1.012
## lambda        0.999      1.000
##
## Multivariate psrf
##
## 1.01
##
## ********** The Heidelberger-Welch diagnostic: **********
##
## Chain 1, epsilon=0.1, alpha=0.05
```

```
##          Stationarity start     p-value
##          test         iteration
## b[1]     passed       1         0.1016
## b[10]    passed       1         0.0989
## b[2]     passed       1         0.1117
## b[3]     passed       1         0.7080
## b[4]     passed       1         0.8546
## b[5]     passed       1         0.3624
## b[6]     passed       1         0.6680
## b[7]     passed       1         0.9542
## b[8]     passed       1         0.9951
## b[9]     passed       1         0.3240
## deviance passed       1         0.2433
## int      passed       1         0.2431
## lambda   passed       1         0.6522
##
##          Halfwidth Mean      Halfwidth
##          test
## b[1]     passed      -0.0848 0.00371
## b[10]    passed       0.0889 0.00368
## b[2]     passed      -0.7072 0.00311
## b[3]     passed       0.7431 0.00292
## b[4]     passed      -0.1975 0.00249
## b[5]     passed       0.0860 0.00345
## b[6]     passed       1.2165 0.00340
## b[7]     passed       0.6488 0.00690
## b[8]     passed      -1.0967 0.00794
## b[9]     passed      -0.0407 0.00324
## deviance passed    6737.8917 0.29978
## int      passed      -1.9741 0.00329
## lambda   passed       2.3039 0.06436
##
## Chain 2, epsilon=0.1, alpha=0.005
##          Stationarity start     p-value
##          test         iteration
## b[1]     passed       1         0.3316
## b[10]    passed       1         0.2549
## b[2]     passed       1         0.3217
## b[3]     passed       1         0.0755
## b[4]     passed       1         0.2663
## b[5]     passed       1         0.0308
## b[6]     passed       1         0.1320
## b[7]     passed       1         0.9003
## b[8]     passed       1         0.8972
## b[9]     passed       1         0.1930
## deviance passed       1         0.3266
## int      passed       1         0.5330
## lambda   passed       1         0.2214
##
##          Halfwidth Mean      Halfwidth
##          test
## b[1]     passed      -0.0865 0.00406
## b[10]    passed       0.0882 0.00294
## b[2]     passed      -0.7065 0.00316
```

```
## b[3]     passed       0.7415 0.00308
## b[4]     passed      -0.1970 0.00251
## b[5]     passed       0.0825 0.00347
## b[6]     passed       1.2129 0.00392
## b[7]     passed       0.6448 0.00703
## b[8]     passed      -1.0920 0.00757
## b[9]     passed      -0.0390 0.00318
## deviance passed    6738.1529 0.29677
## int      passed      -1.9697 0.00336
## lambda   passed       2.3283 0.06876
##
## Chain 3, epsilon=0.189, alpha=0.05
##          Stationarity start     p-value
##          test         iteration
## b[1]     passed       1         0.5912
## b[10]    passed       1         0.8773
## b[2]     passed       1         0.4119
## b[3]     passed       1         0.3181
## b[4]     passed       1         0.8778
## b[5]     passed       1         0.3049
## b[6]     passed       1         0.0634
## b[7]     passed       1         0.5195
## b[8]     passed       1         0.8419
## b[9]     passed       1         0.4995
## deviance passed       1         0.0862
## int      passed       1         0.2985
## lambda   passed       1         0.3681
##
##          Halfwidth Mean      Halfwidth
##          test
## b[1]     passed      -0.0885 0.00331
## b[10]    passed       0.0892 0.00316
## b[2]     passed      -0.7069 0.00294
## b[3]     passed       0.7418 0.00292
## b[4]     passed      -0.1975 0.00241
## b[5]     passed       0.0883 0.00361
## b[6]     passed       1.2133 0.00375
## b[7]     passed       0.6384 0.00722
## b[8]     passed      -1.0885 0.00770
## b[9]     passed      -0.0398 0.00348
## deviance passed    6738.1443 0.32008
## int      passed      -1.9751 0.00342
## lambda   passed       2.3258 0.06571
##
## ********** The Raftery-Lewis diagnostic: **********
##
## Chain 1, converge.eps = 0.001
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
##
## Chain 2, converge.eps = 0.001
```

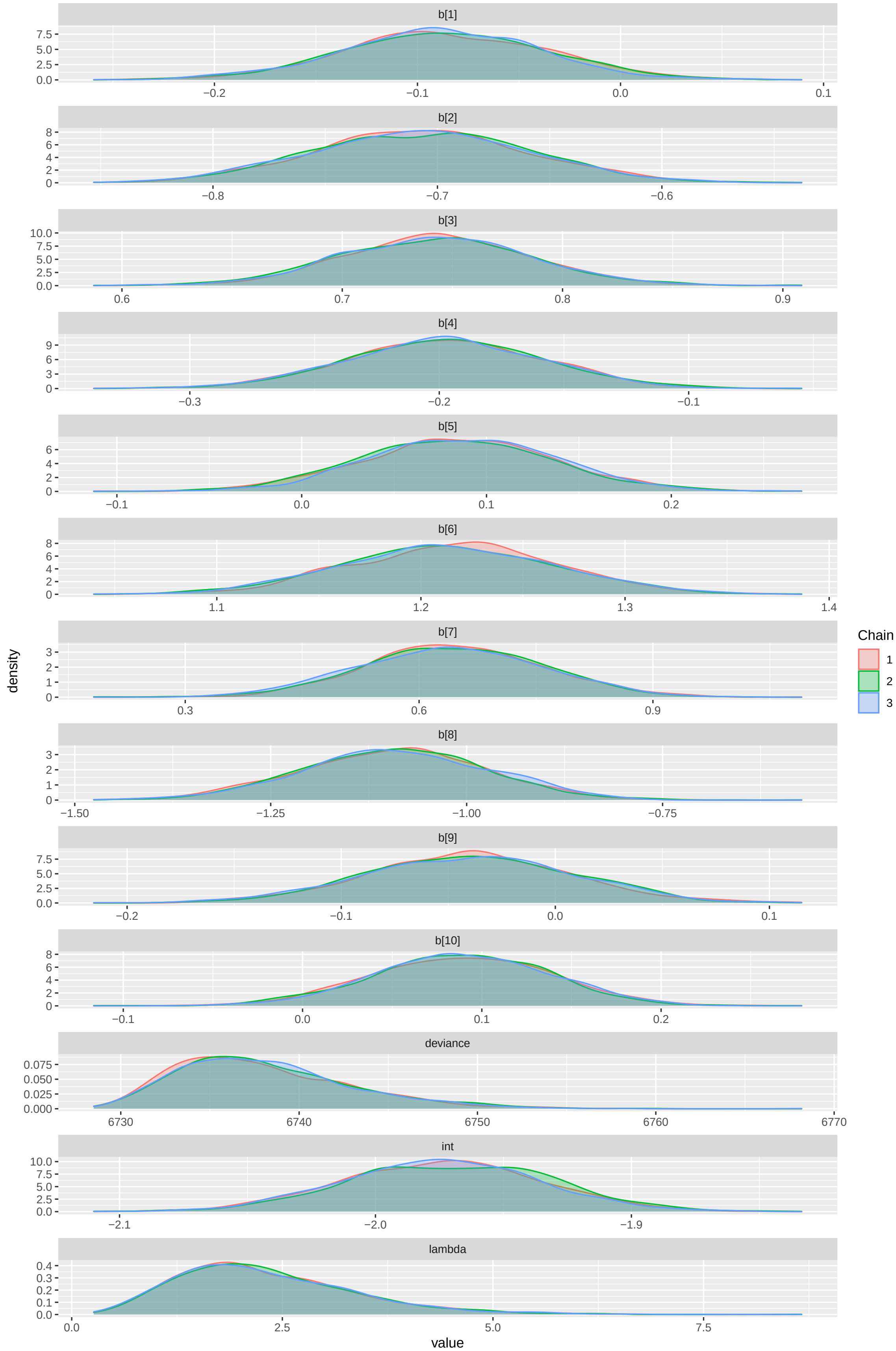
```
## Quantile (q) = 0.1
## Accuracy (r) = +/- 0.001
## Probability (s) = 0.9
##
## You need a sample size of at least 243499 with these values of q, r and s
##
## Chain 3, converge.eps = 0.0025
## Quantile (q) = 0.001
## Accuracy (r) = +/- 0.001
## Probability (s) = 0.999
##
## You need a sample size of at least 10817 with these values of q, r and s
```

- **Gelman & Rubin:** To detect whether they've hit the target distribution. We are looking for a value near 1 (and at the very least less than 1.1).
- **Geweke** All chains have converged to a stationary distribution.
- **Raftery**
- **Heidelberg-Welch** This diagnostic tells us what we already know, all of the variables listed previously haven't passed the halfwidth mean test, this means that for convergence to occur we need to extend the iterations.

Density functions MCMC

Double check that the variables that we pointed out are concentrated on 0.

```r
# All in one diagnostics
bayes.mod.fit.gg <- ggs(mod1_sim)
ggs_density(bayes.mod.fit.gg)
```



b[1]
b[2]
b[3]
b[4]
b[5]
b[6]
b[7]
b[8]
b[9]
b[10]
deviance
int
lambda
density
value
Chain
1
2
3

Autocorrelation

```r
# As noted previously, each estimate in the MCMC process is serially correlated
# with the previous estimates by definition. Higher serial correlation typically has the effect of requ
# more samples in order to get to a stationary distribution.
autocorr.diag(mod1_sim)
```

```
##                  b[1]        b[10]       b[2]         b[3]         b[4]
## Lag 0    1.000000000  1.000000000 1.00000000  1.000000000  1.000000000
## Lag 4    0.132529391  0.062641592 0.05170723  0.123911778  0.054841449
## Lag 20   0.002128337 -0.015802253 0.01122646 -0.007246245 -0.003067081
## Lag 40   0.025539662  0.000115204 0.01988906 -0.008084708  0.010385504
## Lag 200 -0.028555595  0.001821548 0.01976237 -0.002155188 -0.005104623
##                  b[5]         b[6]         b[7]        b[8]         b[9]
## Lag 0    1.000000000  1.000000000  1.000000000  1.00000000 1.000000000
## Lag 4    0.096366053  0.124304338  0.038544611  0.05327879 0.116454324
## Lag 20  -0.001663751  0.011772125  0.026816492  0.02619700 0.022626015
## Lag 40  -0.001107793 -0.020621138 -0.050378588 -0.03524735 0.026384373
## Lag 200 -0.011291069  0.005637233  0.008457442  0.01326015 0.003594754
##              deviance           int       lambda
## Lag 0    1.0000000000  1.0000000000  1.000000000
## Lag 4    0.0339945492  0.3090754869  0.006931147
## Lag 20   0.0002764613 -0.0037013968 -0.021287309
## Lag 40  -0.0068275003 -0.0338632036  0.017554734
## Lag 200  0.0192348282 -0.0007622598 -0.013874229
```

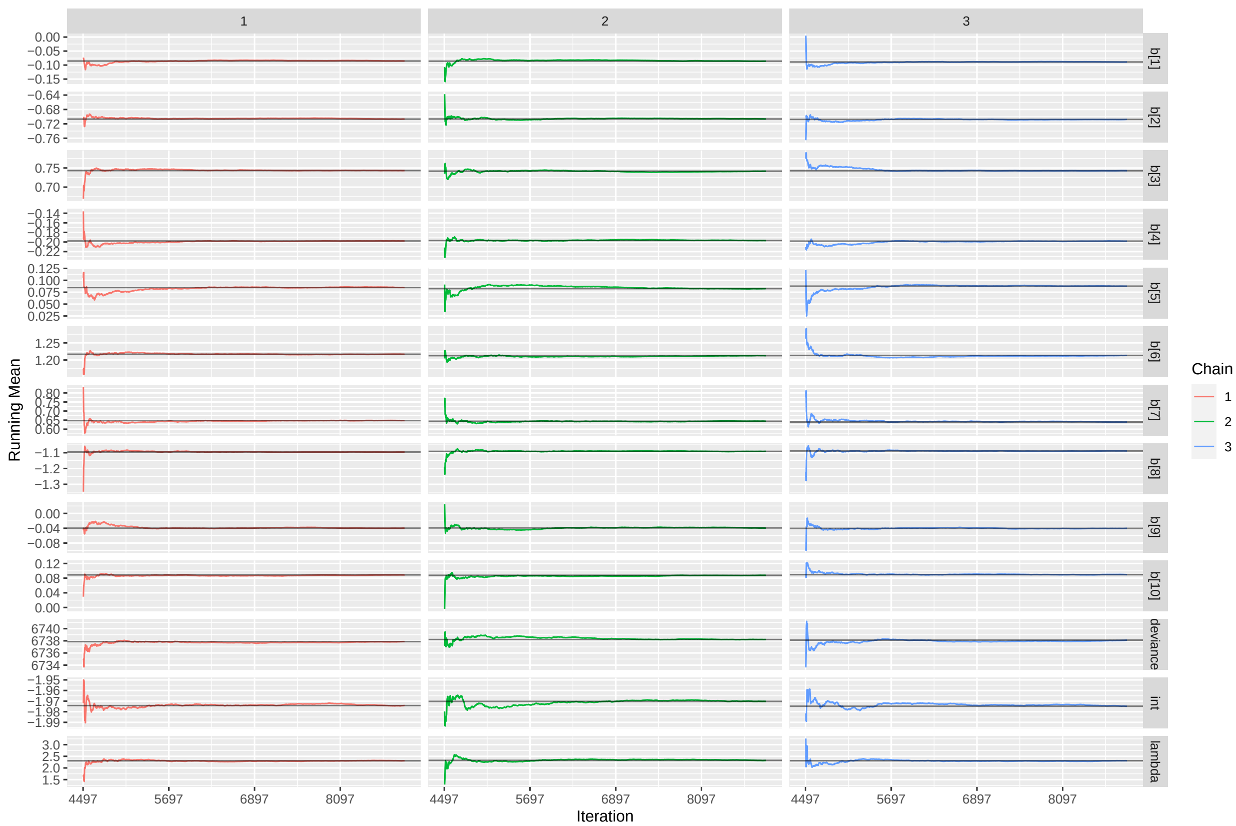
```r
effectiveSize(mod1_sim)
```

```
##     b[1]    b[10]     b[2]     b[3]     b[4]     b[5]     b[6]     b[7]
## 2588.021 2876.397 3109.931 2556.201 3113.469 2783.921 2527.982 3277.718
##     b[8]     b[9] deviance      int   lambda
## 2933.383 2670.183 3208.994 1726.211 3245.023
```

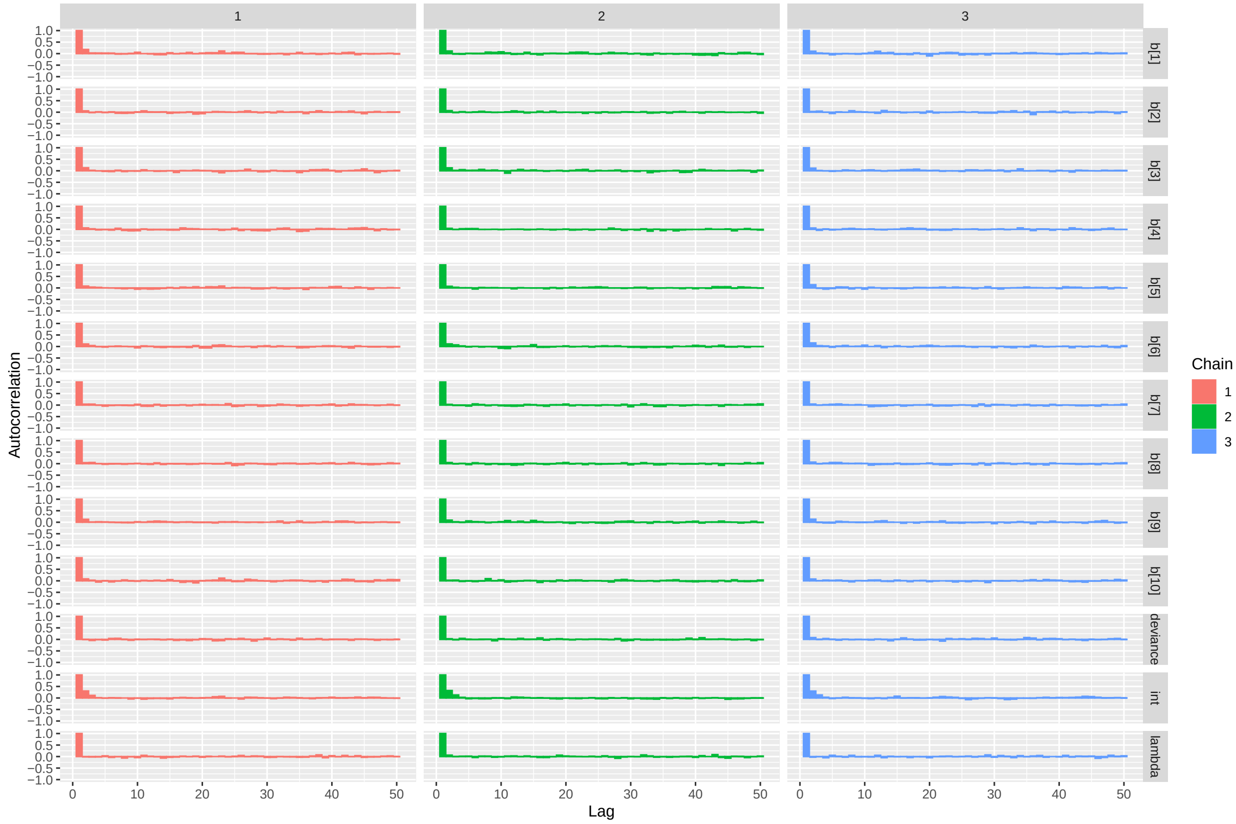
```r
mod1$BUGSoutput$DIC
```

```
## [1] 6749.87
```

```r
results1=ggs(mod1_sim)
ggs_running(results1)
```




```r
ggs_autocorrelation(results1)
```



5. Model n.2 using Double Exponential prior (LASSO)

Now we are going to create a prior using the double exponential probabilty distribution. Not only that, but we are going to use only the variables that are significant, this means we'll delete the ones that were distributed with a mean around 0. The variables that have been dropped are:

- Humidity3pm (Beta 7)
- Pressure9am (Beta 8)
- Pressure3pm (Beta 9)
- Cloud9am (Beta 10)
- Temp3pm (Beta 11)

```
# Let's write down the model
inits_0 = list("int" = 0.2, 'lambda' = 0.2, 'b' = rep(0.1,5))
inits_1 = list("int" = 0.15, 'lambda' = 0.15, 'b' = rep(0.05,5))
inits_2 = list("int" = 0.25, 'lambda' = 0.25, 'b' = rep(0.2,5))
inits_total2 = list(inits_0,inits_1,inits_2)

# Variables
data_jags2 = list(y = new_data$RainTomorrow,
                  Evaporation = X[,4],
                  Sunshine = X[,5],
                  WindGustSpeed = X[,6],
                  WindSpeed9am = X[,7],
                  Humidity9am = X[,9])
```

```r
mod2_string = "model{
  for (i in 1:length(y)){

y[i] ~ dbern(p[i])

logit(p[i])= int +
    b[1]*Evaporation[i] + b[2]*Sunshine[i] + b[3]*WindGustSpeed[i] + b[4]*WindSpeed9am[i] + b[5]*Humidi
  }

int ~ dnorm(0.0,1.0E-6)

for (j in 1:5){
b[j] ~ ddexp(0.0,lambda)
}
lambda ~ dgamma(0.1,0.1)
}"



params = c('int','b','lambda')
mod2 = jags(data = data_jags2, inits = inits_total2,
            parameters.to.save = params,model.file = textConnection(mod2_string),n.chains = 3,
            n.iter = 9000)
```

```
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 10000
##    Unobserved stochastic nodes: 7
##    Total graph size: 80497
##
## Initializing model
```

```r
mod2_sim = as.mcmc(mod2)
options(scipen=999)
```

```r
mod2$BUGSoutput$summary
```

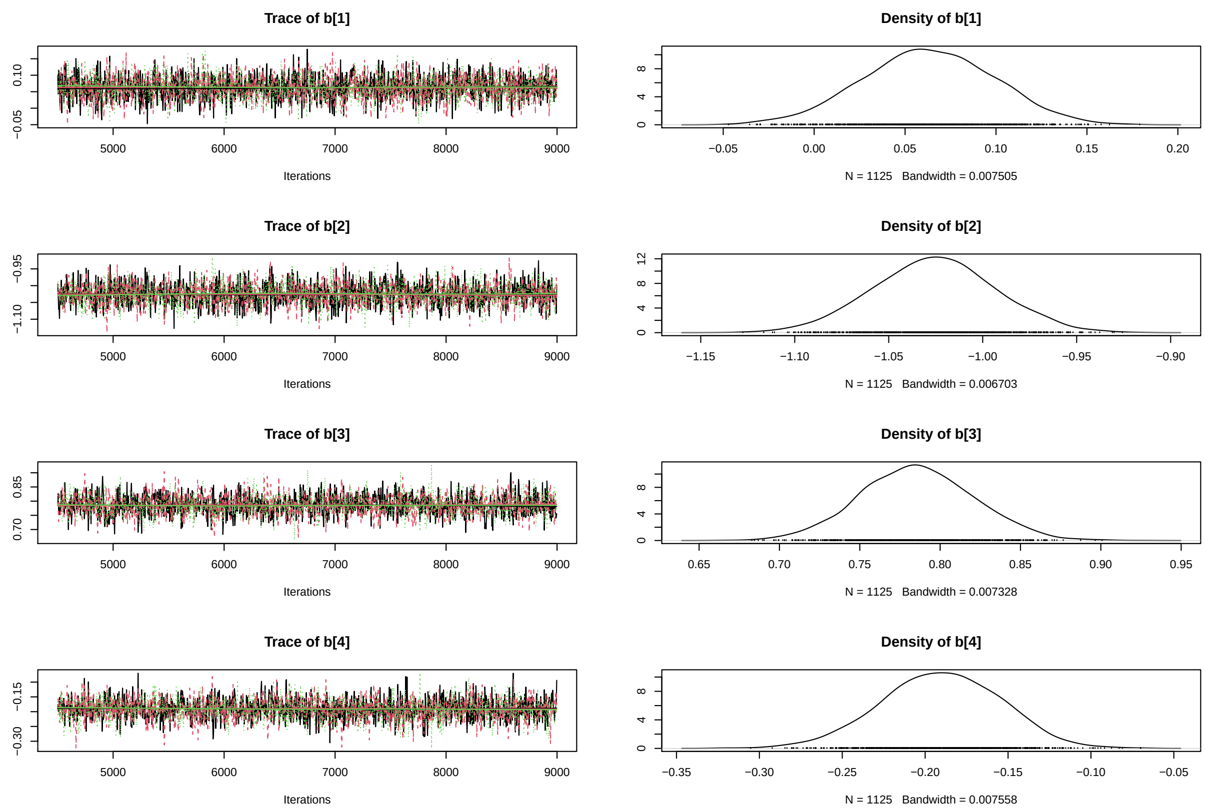
```
##                   mean         sd          2.5%          25%          50%
## b[1]        0.06285212 0.03595114   -0.007922125    0.03875624    0.0627049
## b[2]       -1.02567120 0.03257623   -1.089956323   -1.04745232   -1.0256304
## b[3]        0.78576131 0.03534146    0.717929248    0.76177948    0.7853048
## b[4]       -0.19065027 0.03620629   -0.262690418   -0.21477696   -0.1902421
## b[5]        0.54809077 0.04062500    0.468649519    0.52055976    0.5484953
## deviance 7708.27431052 3.47826344 7703.528444223 7705.70794374 7707.6061358
## int        -1.77477597 0.03449328   -1.844938249   -1.79839008   -1.7746886
## lambda      1.87915168 0.82267733    0.638005040    1.28302830    1.7499194
##                  75%        97.5%     Rhat n.eff
## b[1]        0.0872476    0.1322078 1.001253  2900
## b[2]       -1.0044247   -0.9624331 1.001146  3400
## b[3]        0.8088175    0.8562275 1.000756  3400
```

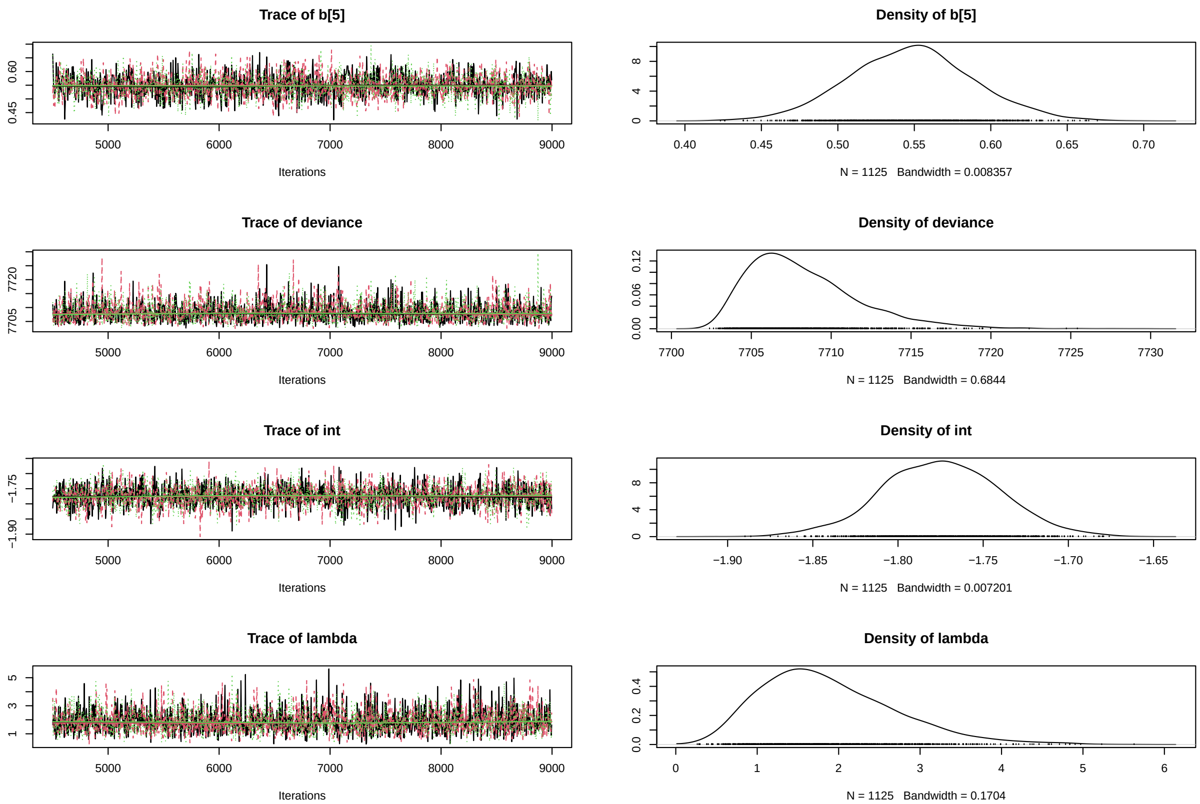
```
## b[4]        -0.1652669   -0.1229784 1.000742  3400
## b[5]         0.5742029    0.6303224 1.001083  3400
## deviance 7710.1006619 7716.8007591 1.001667  1900
## int         -1.7514611   -1.7066800 1.000974  3400
## lambda       2.3766354    3.7877398 1.000900  3400
```

Trace & Density plots

Once we've dropped the variables that were not explanatory we can see that the rest are valuable and all tend to converge to a stationary distribution (graphically seen in the traceplot)

```
plot.new()
par(mfrow = c(3,4))
plot(mod2_sim)
```





Diagnostics

```r
superdiag(mod2_sim, burnin = 100)
```

```
## Number of chains = 3
## Number of iterations = 1125 per chain before discarding the burn-in period
## The burn-in period = 100 per chain
## Sample size in total = 3075
##
## ********** The Geweke diagnostic: **********
## Z-scores:
##                        chain1     chain 2     chain 3
## b[1]                -1.3657273  0.07244666  0.70232562
## b[2]                 1.3181270  0.93647662  0.07399363
## b[3]                -1.0770809 -0.67643012 -1.37629066
## b[4]                 0.7363227  0.17772045  1.90465338
## b[5]                -1.2339605 -0.30048399  0.67373953
## deviance             1.4102252  2.16913576 -0.12136441
## int                  2.6096379 -0.06528719 -0.38079454
## lambda               0.2633492 -1.39620727 -0.88659682
## Window From Start    0.1000000  0.61945000  0.62720000
## Window From Stop     0.5000000  0.22964000  0.10458000
##
## ********** The Gelman-Rubin diagnostic: **********
## Potential scale reduction factors:
```

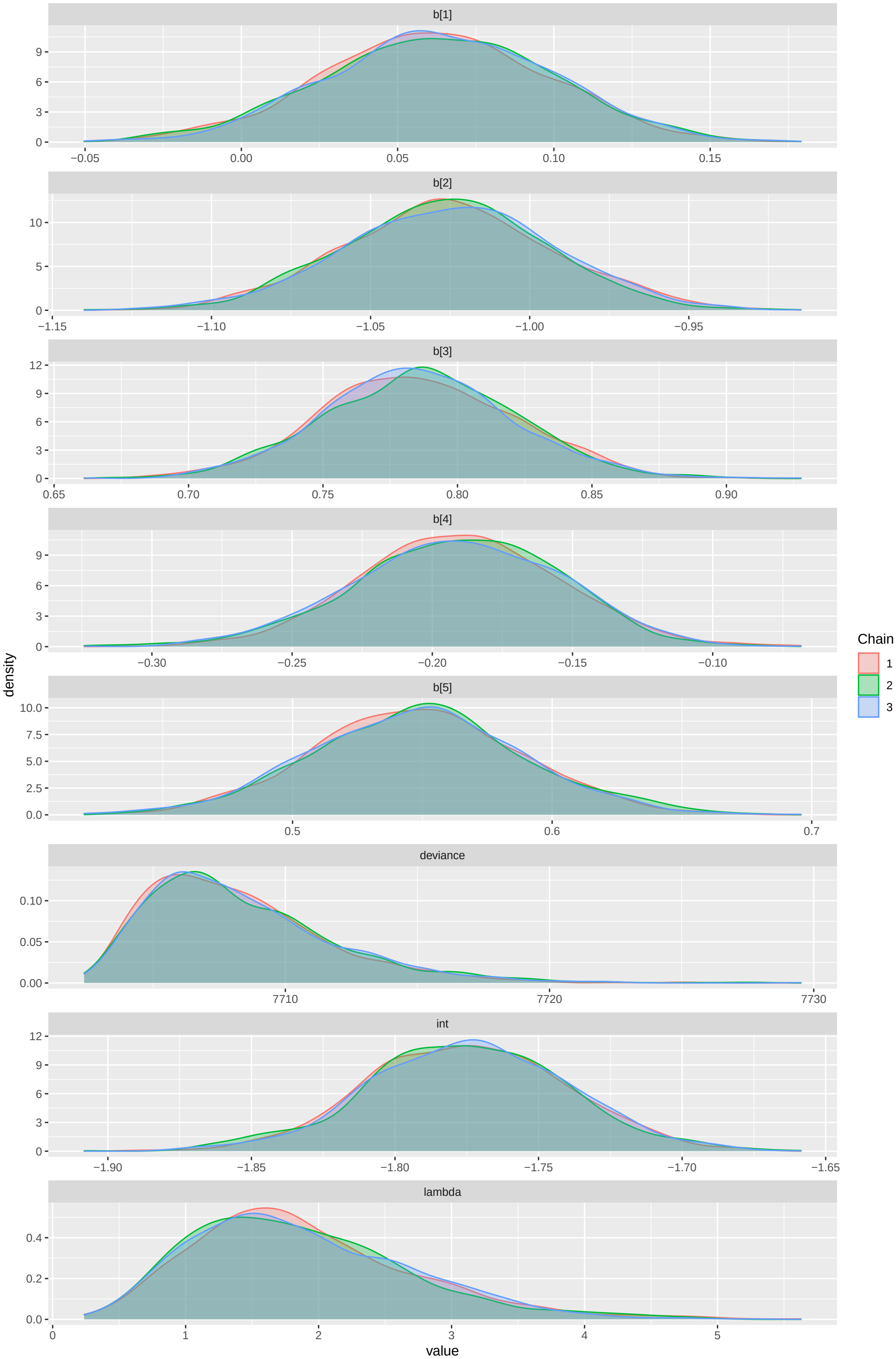
```
##
##          Point est. Upper C.I.
## b[1]          1.001      1.003
## b[2]          1.001      1.003
## b[3]          0.999      0.999
## b[4]          1.000      1.001
## b[5]          1.003      1.006
## deviance      1.000      1.002
## int           1.001      1.005
## lambda        1.004      1.006
##
## Multivariate psrf
##
## 1
##
## ********** The Heidelberger-Welch diagnostic: **********
##
## Chain 1, epsilon=0.1, alpha=0.05
##          Stationarity start     p-value
##          test         iteration
## b[1]     passed       104       0.0924
## b[2]     passed         1       0.5453
## b[3]     passed         1       0.6278
## b[4]     passed         1       0.6304
## b[5]     passed         1       0.0818
## deviance passed         1       0.8332
## int      passed         1       0.2657
## lambda   passed         1       0.2624
##
##          Halfwidth Mean      Halfwidth
##          test
## b[1]     passed       0.0623 0.00227
## b[2]     passed      -1.0253 0.00213
## b[3]     passed       0.7851 0.00255
## b[4]     passed      -0.1901 0.00242
## b[5]     passed       0.5475 0.00306
## deviance passed    7708.1266 0.22426
## int      passed      -1.7747 0.00241
## lambda   passed       1.9013 0.05139
##
## Chain 2, epsilon=0.169, alpha=0.025
##          Stationarity start     p-value
##          test         iteration
## b[1]     passed       1         0.376
## b[2]     passed       1         0.485
## b[3]     passed       1         0.584
## b[4]     passed       1         0.570
## b[5]     passed       1         0.549
## deviance passed       1         0.052
## int      passed       1         0.725
## lambda   passed       1         0.290
##
##          Halfwidth Mean      Halfwidth
##          test
```

```
## b[1]     passed       0.0631 0.00241
## b[2]     passed      -1.0268 0.00209
## b[3]     passed       0.7865 0.00256
## b[4]     passed      -0.1915 0.00242
## b[5]     passed       0.5493 0.00272
## deviance passed    7708.4169 0.24199
## int      passed      -1.7749 0.00231
## lambda   passed       1.8662 0.05076
##
## Chain 3, epsilon=0.133, alpha=0.005
##          Stationarity start     p-value
##          test         iteration
## b[1]     passed       1         0.656
## b[2]     passed       1         0.526
## b[3]     passed       1         0.262
## b[4]     passed       1         0.276
## b[5]     passed       1         0.903
## deviance passed       1         0.826
## int      passed       1         0.948
## lambda   passed       1         0.666
##
##          Halfwidth Mean      Halfwidth
##          test
## b[1]     passed       0.0636 0.00231
## b[2]     passed      -1.0243 0.00213
## b[3]     passed       0.7849 0.00257
## b[4]     passed      -0.1911 0.00262
## b[5]     passed       0.5470 0.00274
## deviance passed    7708.3309 0.21244
## int      passed      -1.7735 0.00210
## lambda   passed       1.8710 0.04928
##
## ********** The Raftery-Lewis diagnostic: **********
##
## Chain 1, converge.eps = 0.001
## Quantile (q) = 0.025
## Accuracy (r) = +/- 0.005
## Probability (s) = 0.95
##
## You need a sample size of at least 3746 with these values of q, r and s
##
## Chain 2, converge.eps = 0.0002
## Quantile (q) = 0.25
## Accuracy (r) = +/- 0.0005
## Probability (s) = 0.95
##
## You need a sample size of at least 2881095 with these values of q, r and s
##
## Chain 3, converge.eps = 0.001
## Quantile (q) = 0.01
## Accuracy (r) = +/- 0.001
## Probability (s) = 0.9
##
## You need a sample size of at least 26785 with these values of q, r and s
```

All of the diagnostics give us the same result, convergence to a stationary distribution. In this case even the Heidelberg-Welch diagnostics gives us positive outcomes for both Stationary start and Halfwidth Mean.

Density functions MCMC

```r
bayes.mod.fit.gg <- ggs(mod2_sim)
ggs_density(bayes.mod.fit.gg)
```


b[1]
b[2]
b[3]
b[4]
b[5]
deviance
int
lambda
density
value
Chain
1
2
3

Autocorrelation

```r
autocorr.diag(mod2_sim)
```

```
##                 b[1]         b[2]         b[3]          b[4]        b[5]
## Lag 0    1.000000000  1.000000000  1.000000000  1.0000000000  1.00000000
## Lag 4    0.065193401  0.055295002  0.159476185  0.1006231675  0.09902700
## Lag 20  -0.003764338 -0.003011953 -0.025862425 -0.0288354876  0.02426053
## Lag 40   0.011939593 -0.031257078  0.011193558  0.0091459018  0.03111746
## Lag 200  0.005167101 -0.008884855 -0.001957977 -0.0003677772 -0.01574325
##             deviance        int         lambda
## Lag 0   1.000000000 1.00000000  1.00000000000
## Lag 4   0.059340333 0.05659343 -0.01632735764
## Lag 20  0.001454357 0.01124475 -0.00002931683
## Lag 40  0.004498432 0.02370172  0.00725974522
## Lag 200 0.007087569 0.02278779 -0.00380671937
```

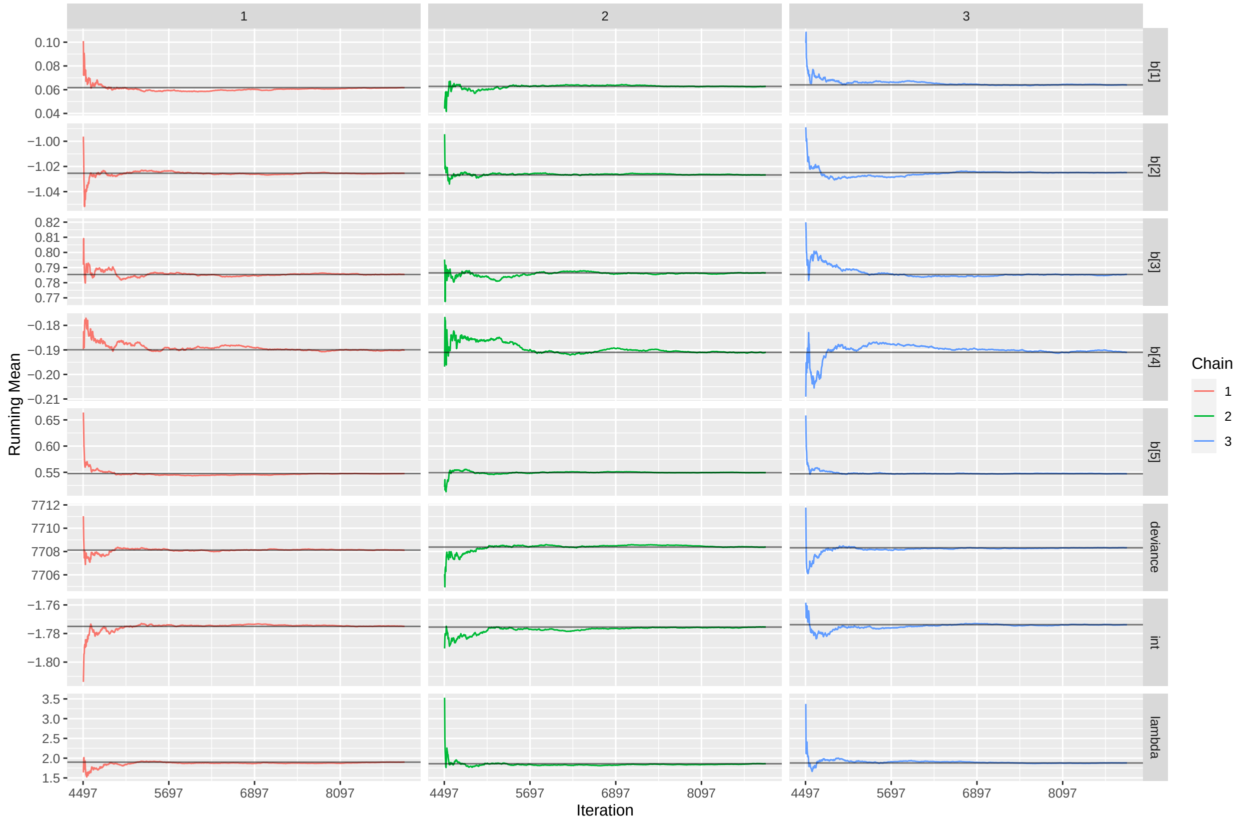
```r
effectiveSize(mod2_sim)
```

```
##     b[1]     b[2]     b[3]     b[4]     b[5] deviance      int   lambda
## 3116.950 3018.631 2445.279 2759.182 2767.965 2999.553 2930.330 3375.000
```

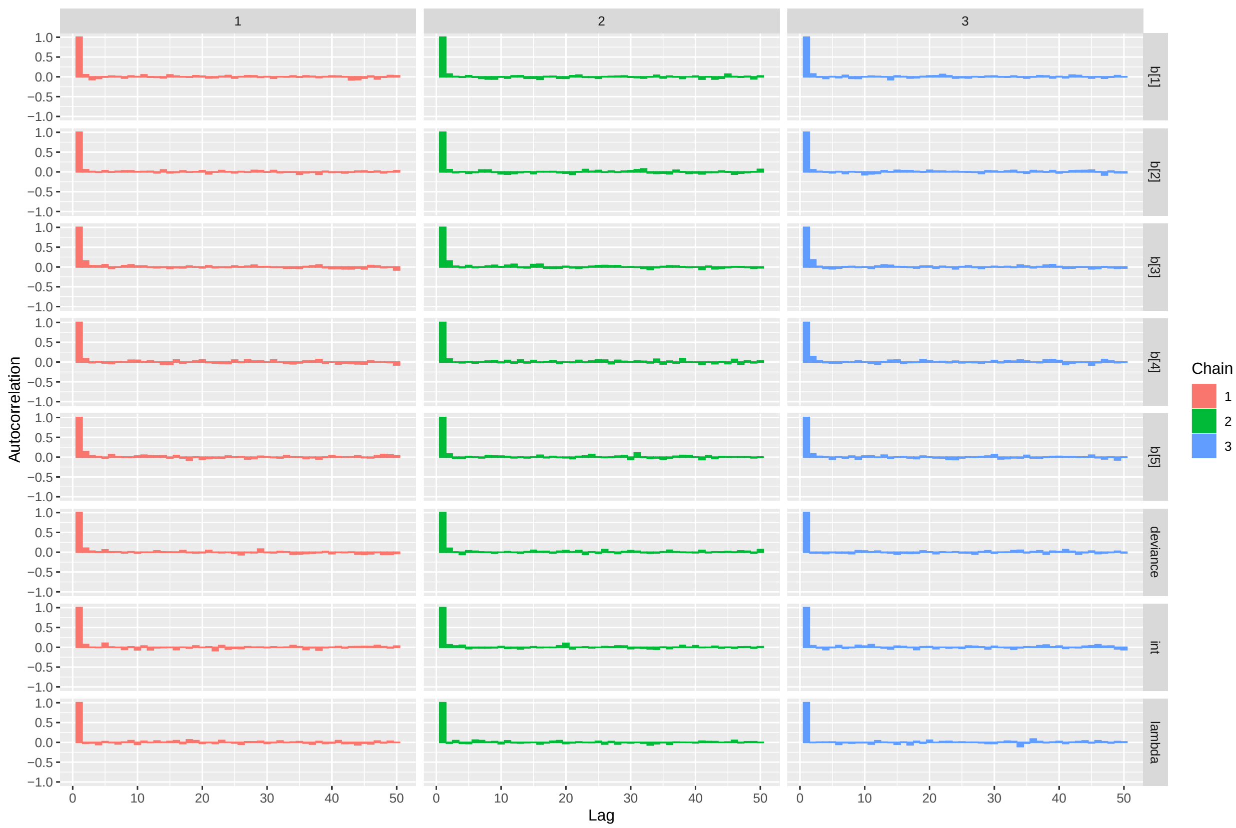
```r
mod2$BUGSoutput$DIC
```

```
## [1] 7714.321
```

```r
results2=ggs(mod2_sim)
ggs_running(results2)
```




```
ggs_autocorrelation(results2)
```



6. Prediction

```
XT = X
```

```
mod_csim = as.mcmc(do.call(rbind,mod1_sim))
posterior_coef = colMeans(mod_csim)
posterior_coef_list = c(posterior_coef[6], posterior_coef[7], posterior_coef[8], posterior_coef[9], pos
predicted = posterior_coef['int'] + XT[,c(4, 5, 6, 7, 9)] %*% posterior_coef_list
p_hat = 1.0/(1.0 + exp(-predicted))
tab0.5a = table(p_hat > 0.4, y)
performance_mod1 = sum(diag(tab0.5a))/sum(tab0.5a)
var_mod1 = mean((p_hat - y)^2)
```

```
mod2_csim = as.mcmc(do.call(rbind,mod2_sim))
mod2_csim =as.matrix(mod2_csim)
posterior_coef2 = colMeans(mod2_csim)
predicted2 = posterior_coef2['int'] + XT[,c(4, 5, 6, 7, 9)] %*% posterior_coef2[1:5]
p_hat2 = 1.0/(1.0 + exp(-predicted2))
tab0.5b = table(p_hat2 > 0.4, y)
performance_mod2 = sum(diag(tab0.5b))/sum(tab0.5b)
var_mod2 = mean((p_hat2 - y)^2)
```

```
predicted.data <- data.frame(
  p = p_hat,
```

```r
  rain = y
)
predicted.data <- predicted.data[
  order(predicted.data$p,decreasing = FALSE),]
predicted.data$rank <- 1:nrow(predicted.data)

predicted.data3 <- data.frame(
  p = p_hat2,
  rain = y
)
predicted.data3 <- predicted.data3[
  order(predicted.data3$p,decreasing = FALSE),]
predicted.data3$rank <- 1:nrow(predicted.data3)

# Frequentist analysis (2, 4, 5, 6, 7, 9)
glm.fit <- glm(y ~ X[,4] + X[,5]+X[,6]+X[,7]+X[,9],
               family = 'binomial'(link="logit"), maxit = 50)
### The model doesn't have Rain Tomorrow variable
summary(glm.fit)
```

```
##
## Call:
## glm(formula = y ~ X[, 4] + X[, 5] + X[, 6] + X[, 7] + X[, 9],
##     family = binomial(link = "logit"), maxit = 50)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.4135  -0.5873  -0.3426  -0.1798   2.9293
##
## Coefficients:
##             Estimate Std. Error z value             Pr(>|z|)
## (Intercept) -1.77648    0.03502 -50.721 < 0.0000000000000002 ***
## X[, 4]       0.06750    0.03623   1.863               0.0624 .
## X[, 5]      -1.02695    0.03287 -31.247 < 0.0000000000000002 ***
## X[, 6]       0.78926    0.03542  22.280 < 0.0000000000000002 ***
## X[, 7]      -0.19377    0.03505  -5.529         0.0000000323 ***
## X[, 9]       0.55147    0.03986  13.836 < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 10523.0  on 9999  degrees of freedom
## Residual deviance:  7702.2  on 9994  degrees of freedom
## AIC: 7714.2
##
## Number of Fisher Scoring iterations: 5
```

```r
# Frequentist prediction
coef_freq = summary(glm.fit)$coefficients[,1]
freq_predicted = coef_freq ['(Intercept)'] + XT[,c(4, 5, 6, 7, 9)] %*% coef_freq [2:6]
p_hat_freq = 1.0/(1.0 + exp(-freq_predicted))
tab0.5c = table(p_hat_freq > 0.5, y)
```

```r
performance_freq = sum(diag(tab0.5c))/sum(tab0.5c)
var_freq = mean((p_hat_freq - y)^2)
predicted.data2 <- data.frame(
  p = p_hat_freq,
  rain = y
)
predicted.data2 <- predicted.data2[
  order(predicted.data2$p,decreasing = FALSE),]
predicted.data2$rank <- 1:nrow(predicted.data2)

# Main results

print(tab0.5a)
```

```
##        y
##            0    1
##   FALSE 4183  521
##   TRUE  3623 1673
```

```r
print(tab0.5b)
```

```
##        y
##            0    1
##   FALSE 6982  981
##   TRUE   824 1213
```

```r
print(tab0.5c)
```

```
##        y
##            0    1
##   FALSE 7336 1240
##   TRUE   470  954
```

```r
cat('variance of model 1:',var_mod1,
    'variance of model 2:',var_mod2,
    'variance of model freq:',var_freq,fill = 2)
```

```
## variance of model 1:
## 0.4144
## variance of model 2:
## 0.1217631
## variance of model freq:
## 0.1217597
```

```r
cat('performance of model 1:',performance_mod1,
    'performance of model 2:',performance_mod2,
    'performance of model freq:',performance_freq,fill = 2)
```

```
## performance of model 1:
## 0.5856
```

```
## performance of model 2:
## 0.8195
## performance of model freq:
## 0.829
```

```r
print(mod1$BUGSoutput$DIC)
```

```
## [1] 6749.87
```

```r
print(mod2$BUGSoutput$DIC)
```

```
## [1] 7714.321
```

```r
print(extractAIC(glm.fit))
```

```
## [1]    6.000 7714.231
```

7.Models comparison

A good model comparison criterion to choose may be the DIC (Deviance Information Criterion). The DIC penalty term is based on a complexity term that measures the difference between the expected log likelihood and the log likelihood at the posterior mean point. It is designed specifically for Bayesian estimation that involves MCMC simulations.

DIC is based on the deviance statistics

$$C = logf^*(y|\theta^*) \ D(\theta) = -2logf(y|\theta) + C$$

where: $f(.|.)$ is the likelihood function of the model and $f^*(y|\theta^*)$ is the likelihood of the full model that fits data perfectly.

Because C is constant across models fit to the same data, it is ignored in the actual calculation of DIC.

In the end, the DIC is a sum of two components: the goodness-of-fit term $D(\theta)$ and the model complexity term p_D

$$DIC = D(\bar{\theta}) + 2p_D$$

where:

- $D(\bar{\theta}) = D[E(\theta)]$;
- $p_D = E_\theta[D(\theta)] - D(E[\theta])$

p_D is the complexity of the model (equivalent to the effective number of parameters) and it is defined as the difference between the expected deviance (the larger it is, the worse is the fit) and the deviance of fitted values. That is, the more complex the model is, the larger p_D will be and that is a sign of overfitting.

So, since lower deviance means a model that better fits the data, models with smaller DIC should be preferred to models with larger DIC.

The DIC is easily computed from samples generated by a Monte Carlo Markov Chain simulation and we already have it from the previous outputs:

```r
print(c("Model 1:",mod1$BUGSoutput$DIC))
```

```
## [1] "Model 1:"        "6749.8695292786"
```

```r
print(c("Model 2:",mod2$BUGSoutput$DIC))
```

```
## [1] "Model 2:"         "7714.32079164349"
```

Hence, the first model seems to be the best one since the DIC value is lower and, as we could see previously, it also converges with less iterations with respect to the first model. Thus, we are going to prefer this one.