Solutions to Midterm 2

Shashank Sule

07/11/2019

```
## Loading required package: devtools
## Loading required package: usethis
## Loading required package: ggplot2
## Loading required package: gridExtra
## Loading required package: ProbBayes
## Loading required package: LearnBayes
## Loading required package: dplyr
##
## Attaching package: 'dplyr'
## The following object is masked from 'package:gridExtra':
##
##     combine
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
## Loading required package: shiny
## Loading required package: tidyverse
## -- Attaching packages ---------------------------------------------- tidyverse 1.2.
## v tibble  2.1.3     v purrr   0.3.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ------------------------------------------------- tidyverse_conflicts(
## x dplyr::combine() masks gridExtra::combine()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## Loading required package: runjags
```

```
##
## Attaching package: 'runjags'

## The following object is masked from 'package:tidyr':
##
##     extract

## Loading required package: coda
```

Problem 1

a) Let the ith rating for the jth movie be modelled by $Y_{ij} \sim N(\mu_j, \sigma)$ where μ_j is the movie-specific mean and σ is the universal standard deviation. The standard deviation for Y_{ij} basically captures how much a movie divides opinion. I believe that animation films divide opinion by roughly the same amount. If we had movies from different genres, say thriller or drama, then I would go for a genre-specific deviation. Furthermore, the standard deviation of the rating of each movie reveals the following:

```
## Parsed with column specification:
## cols(
##   userId = col_double(),
##   movieId = col_double(),
##   rating = col_double(),
##   timestamp = col_double(),
##   title = col_character(),
##   Group_Number = col_double()
## )
```

```
##                                                                Deviations
## How to Train Your Dragon (2010)                                 0.8607608
## Toy Story 3 (2010)                                              0.9639329
## Shrek Forever After (a.k.a. Shrek: The Final Chapter) (2010)    1.3228757
## Despicable Me (2010)                                            0.6180165
## Batman: Under the Red Hood (2010)                                      NA
## Legend of the Guardians: The Owls of Ga'Hoole (2010)                   NA
## Megamind (2010)                                                 1.3149778
## Tangled (2010)                                                  0.8881942
```

Thus, the standard deviations seem to either be bunched up around 0.8 or 1.3. So although the more detailed choice is to assume that there are two deviation groups, I use them all under one deviation group to simplify the model.

The hierarchical model can then be set up as follows:

$$
\begin{aligned}
Y_{ij} &\sim N(\mu_j, \sigma_j) \\
\mu_j &\sim N(\mu, \tau) \\
1/\tau^2 &\sim \text{Gamma}(\alpha_\tau, \beta_\tau) \\
1/\sigma_j^2 &\sim \text{Gamma}(\alpha_\sigma, \beta_\sigma)
\end{aligned}
$$

Now I can run JAGS on this hierarchical model.

```
modelString <-"
model {
## likelihood
for (i in 1:N){
y[i] ~ dnorm(mu_j[groups[i]], invsigma2)
}

## priors
for (j in 1:J){
mu_j[j] ~ dnorm(mu, invtau2)
}
invsigma2 ~ dgamma(a_g, b_g)
sigma <- sqrt(pow(invsigma2, -1))

## hyperpriors
mu ~ dnorm(mu0, 1/g0^2)
invtau2 ~ dgamma(a_t, b_t)
tau <- sqrt(pow(invtau2, -1))
}
"
```

```
initsfunction <- function(chain){
  .RNG.seed <- c(1,2)[chain]
  .RNG.name <- c("base::Super-Duper",
                 "base::Wichmann-Hill")[chain]
  return(list(.RNG.seed=.RNG.seed,
              .RNG.name=.RNG.name))
}
```

```
the_data <- list("y" = y, "groups" = groups, "N" = N, "J" = J,
                 "mu0" = 5, "g0" = 0.25,
                 "a_t" = 1, "b_t" = 1,
                 "a_g" = 1, "b_g" = 1)
```

```
posterior <- run.jags(modelString,
                      n.chains = 1,
                      data = the_data,
```

```
                        monitor = c("mu", "tau", "mu_j", "sigma"),
                        adapt = 1000,
                        burnin = 5000,
                        sample = 5000,
                        thin = 1,
                        inits = initsfunction)
```

```
## Calling the simulation...
## Welcome to JAGS 4.3.0 on Sun Nov 24 22:51:32 2019
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 55
##    Unobserved stochastic nodes: 11
##    Total graph size: 138
## . Reading parameter file inits1.txt
## . Initializing model
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 5000
## -------------------------------------------------| 5000
## ************************************************** 100%
## . . . . . Updating 5000
## -------------------------------------------------| 5000
## ************************************************** 100%
## . . . . Updating 0
## . Deleting model
## .
## Note: the model did not require adaptation
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Calculating summary statistics...

## Warning: Convergence cannot be assessed with only 1 chain

## Finished running the simulation
```

```
summary(posterior)
```

```
##          Lower95    Median Upper95      Mean          SD Mode       MCerr
## mu      4.183230 4.6131500 5.10512 4.6196957 0.23703613   NA 0.004826464
## tau     0.455563 0.8809915 1.52712 0.9283788 0.29650788   NA 0.006017271
```

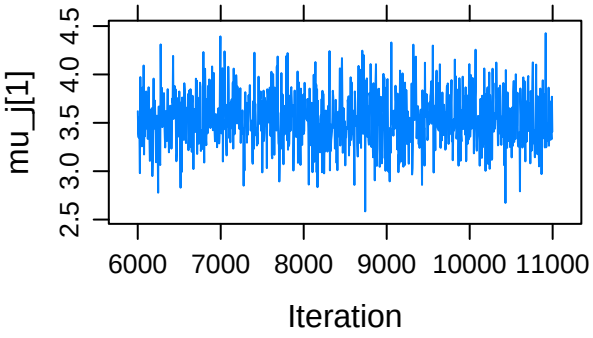
```
## mu_j[1] 2.992000 3.5270000 4.06400 3.5298612 0.27464115   NA 0.004031810
## mu_j[2] 3.417390 3.8743150 4.32395 3.8760266 0.22816050   NA 0.003226677
## mu_j[3] 3.248420 4.1600900 5.06045 4.1651687 0.45824459   NA 0.006861282
## mu_j[4] 3.249570 3.8324950 4.39209 3.8322966 0.29875502   NA 0.004449503
## mu_j[5] 3.521370 4.7452500 6.13826 4.7646193 0.65905855   NA 0.011953718
## mu_j[6] 2.986010 4.3092800 5.59146 4.3024403 0.65923857   NA 0.009558259
## mu_j[7] 2.819560 3.6652850 4.50858 3.6525069 0.42401469   NA 0.006461466
## mu_j[8] 3.679140 4.2416600 4.80394 4.2397929 0.28396058   NA 0.004102672
## sigma   0.755609 0.9252090 1.11641 0.9353367 0.09419858   NA 0.001483166
##         MC%ofSD SSeff        AC.10 psrf
## mu          2.0   2412 -0.003630260   NA
## tau         2.0   2428  0.009461517   NA
## mu_j[1]     1.5   4640 -0.009353859   NA
## mu_j[2]     1.4   5000  0.003926086   NA
## mu_j[3]     1.5   4461 -0.001054741   NA
## mu_j[4]     1.5   4508  0.004472574   NA
## mu_j[5]     1.8   3040  0.013652430   NA
## mu_j[6]     1.4   4757 -0.009463543   NA
## mu_j[7]     1.5   4306 -0.005977131   NA
## mu_j[8]     1.4   4791  0.002997928   NA
## sigma       1.6   4034 -0.003178841   NA
```

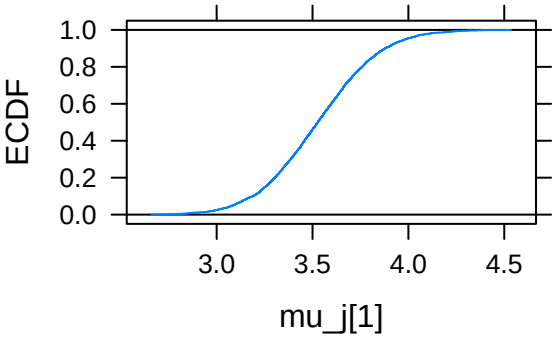
```r
mu_string <- ""

for(i in 1:J){
   mu_string <- c(mu_string, paste("mu_j[",i,"]",sep=""))
}

mu_string <- mu_string[1:J+1]
```

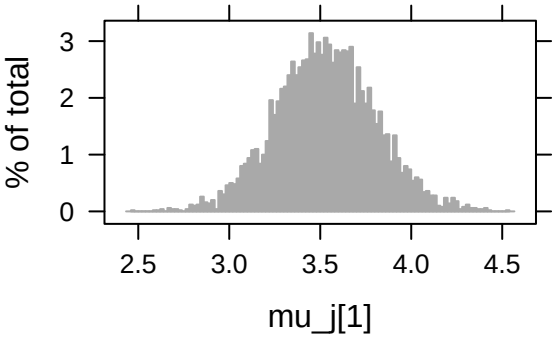
b) I'll plot all of the variables, just to diagnose MCMC convergence

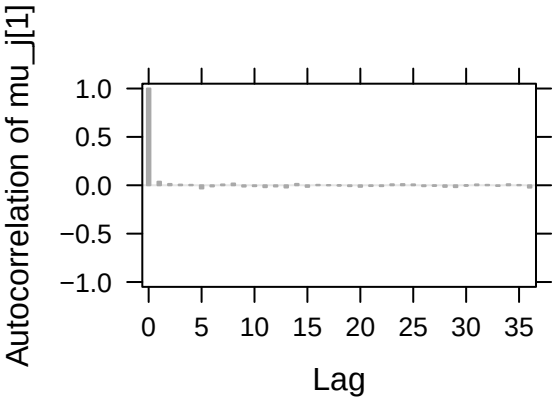
```r
plot(posterior, vars = mu_string)
```

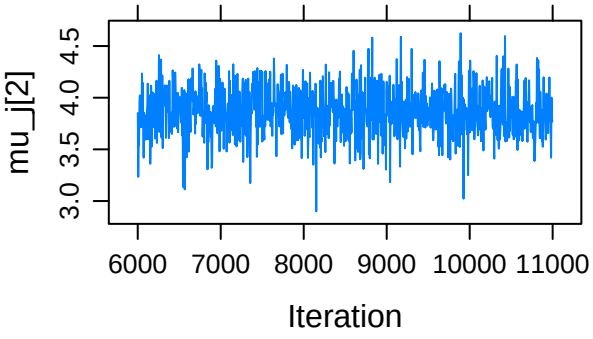
```
## Generating plots...
```

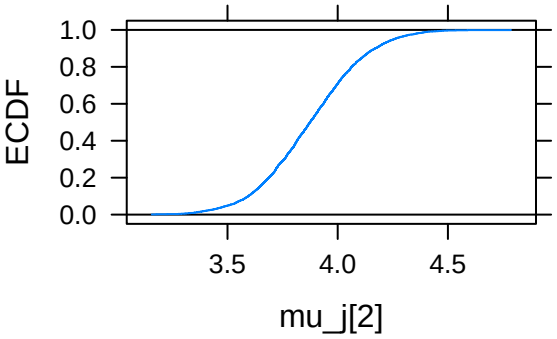

mu_j[1]
2.5 3.0 3.5 4.0 4.5
6000 7000 8000 9000 10000 11000
Iteration

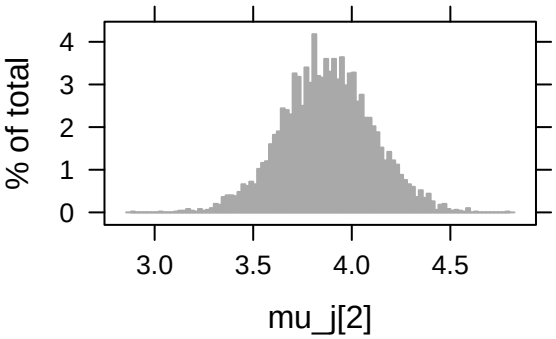

ECDF
0.0 0.2 0.4 0.6 0.8 1.0
3.0 3.5 4.0 4.5
mu_j[1]

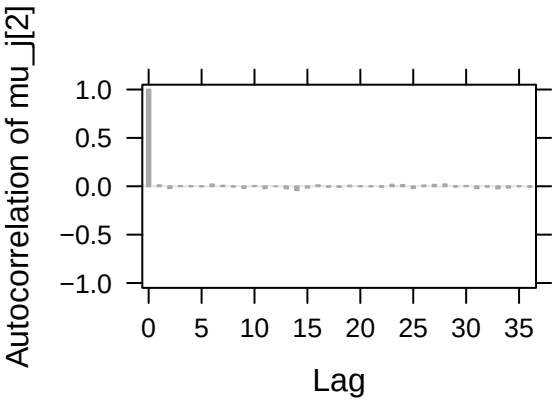

% of total
0 1 2 3
2.5 3.0 3.5 4.0 4.5
mu_j[1]

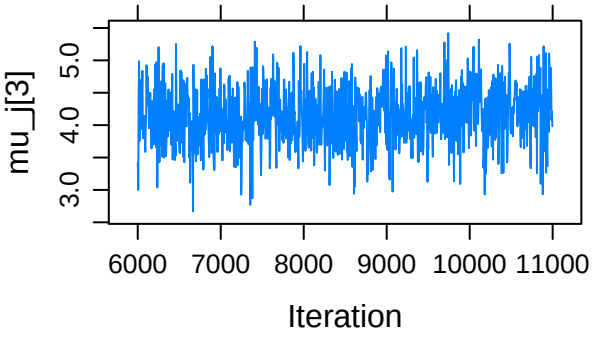

Autocorrelation of mu_j[1]
−1.0 −0.5 0.0 0.5 1.0
0 5 10 15 20 25 30 35
Lag

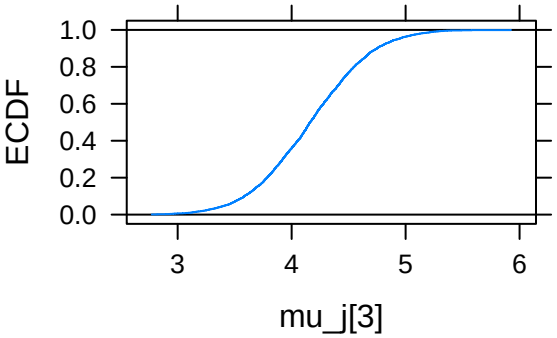

mu_j[2]
3.0 3.5 4.0 4.5
6000 7000 8000 9000 10000 11000
Iteration

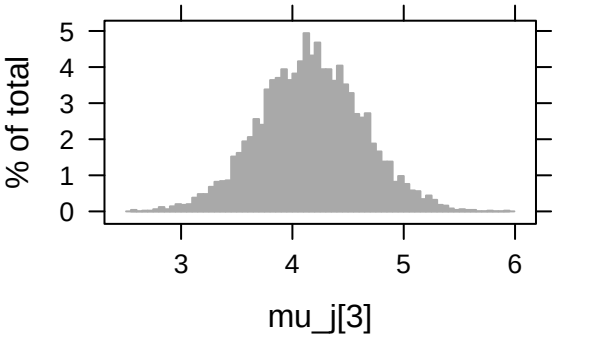

ECDF
0.0 0.2 0.4 0.6 0.8 1.0
3.5 4.0 4.5
mu_j[2]

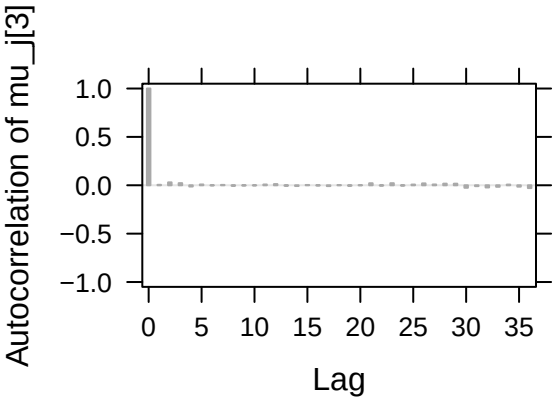

% of total
0 1 2 3 4
3.0 3.5 4.0 4.5
mu_j[2]

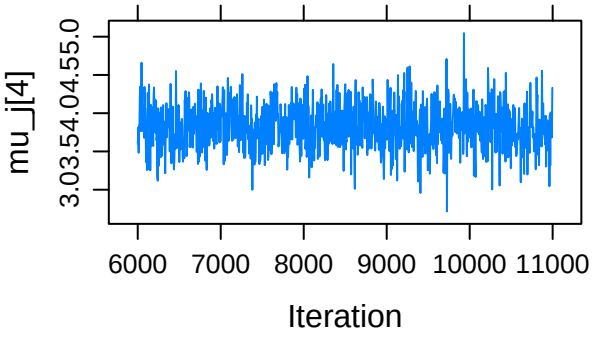

Autocorrelation of mu_j[2]
−1.0 −0.5 0.0 0.5 1.0
0 5 10 15 20 25 30 35
Lag

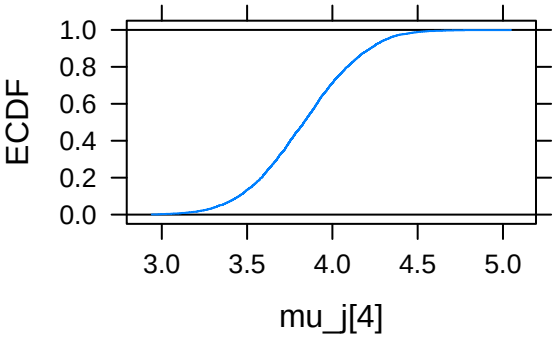

mu_j[3]
3.0
4.0
5.0
6000
7000
8000
9000
10000
11000
Iteration

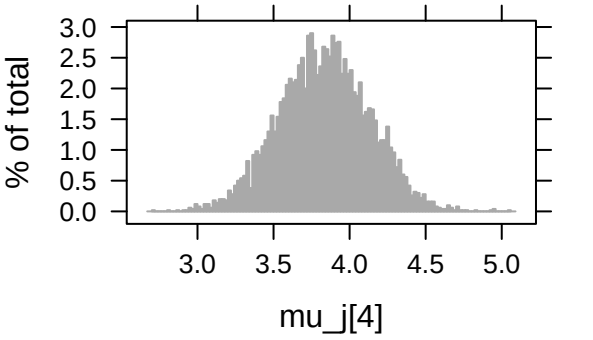

ECDF
0.0
0.2
0.4
0.6
0.8
1.0
3
4
5
6
mu_j[3]

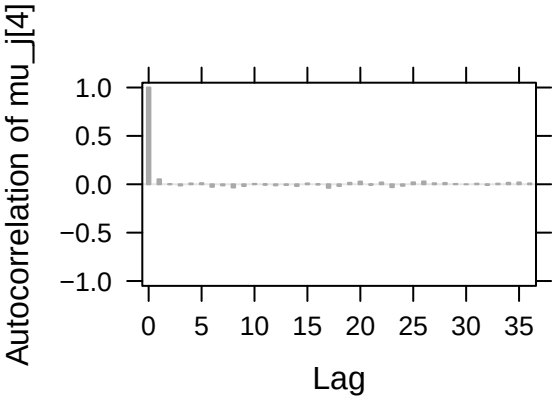

% of total
0
1
2
3
4
5
3
4
5
6
mu_j[3]

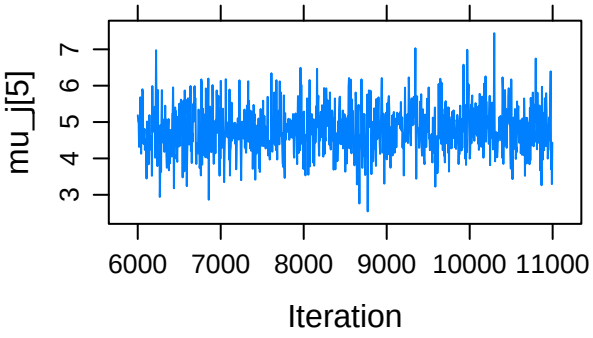

Autocorrelation of mu_j[3]
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
25
30
35
Lag

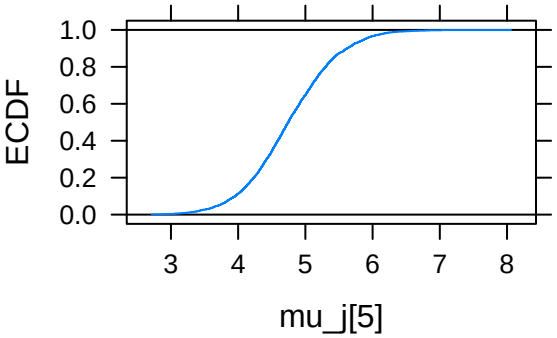

mu_j[4]
3.0
3.5
4.0
4.5
5.0
6000
7000
8000
9000
10000
11000
Iteration

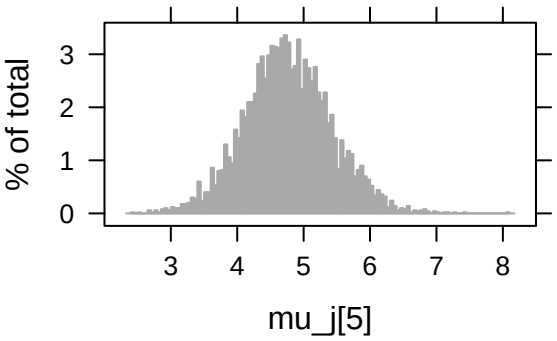

ECDF
0.0
0.2
0.4
0.6
0.8
1.0
3.0
3.5
4.0
4.5
5.0
mu_j[4]

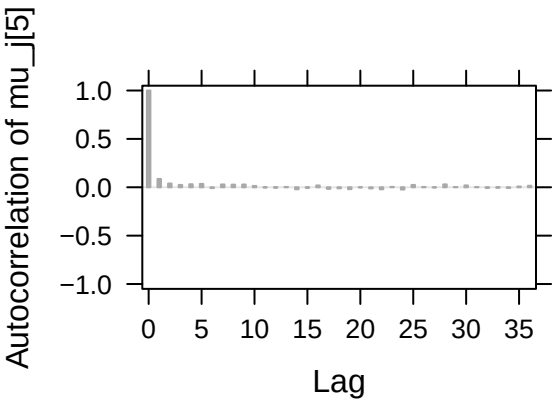

% of total
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.0
3.5
4.0
4.5
5.0
mu_j[4]

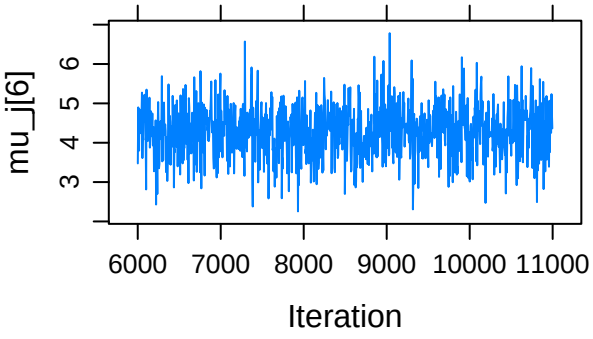

Autocorrelation of mu_j[4]
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
25
30
35
Lag

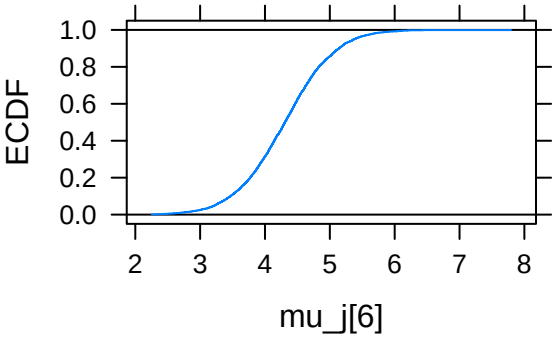

mu_j[5]
3 4 5 6 7
6000 7000 8000 9000 10000 11000
Iteration

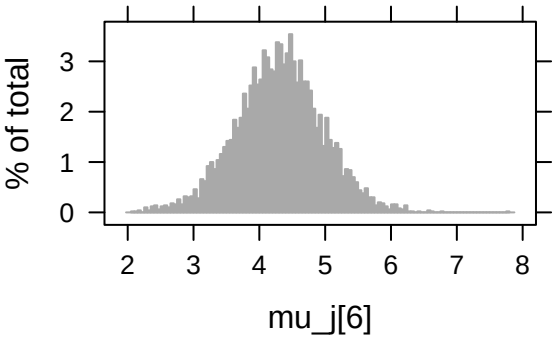

ECDF
0.0 0.2 0.4 0.6 0.8 1.0
3 4 5 6 7 8
mu_j[5]

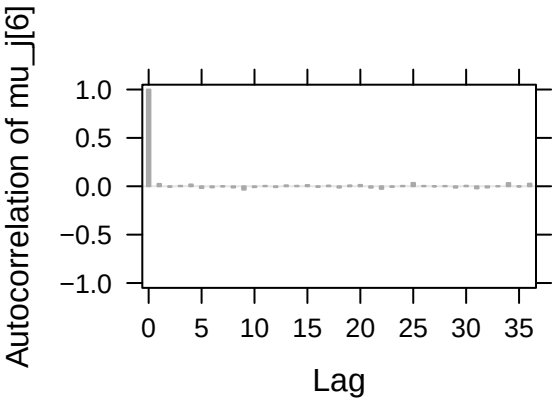

% of total
0 1 2 3
3 4 5 6 7 8
mu_j[5]

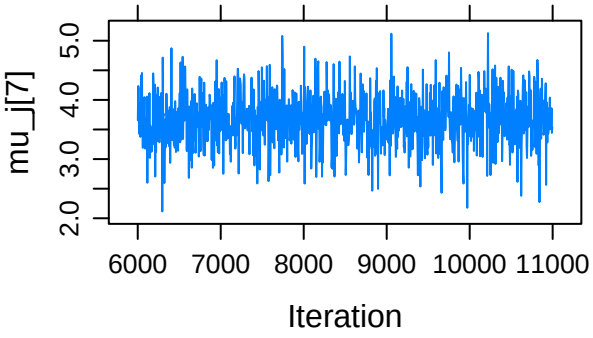

Autocorrelation of mu_j[5]
−1.0 −0.5 0.0 0.5 1.0
0 5 10 15 20 25 30 35
Lag

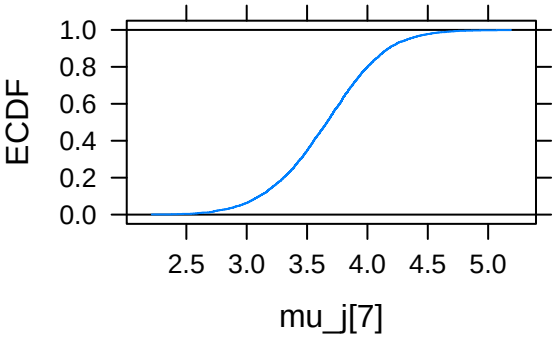

mu_j[6]
3 4 5 6
6000 7000 8000 9000 10000 11000
Iteration

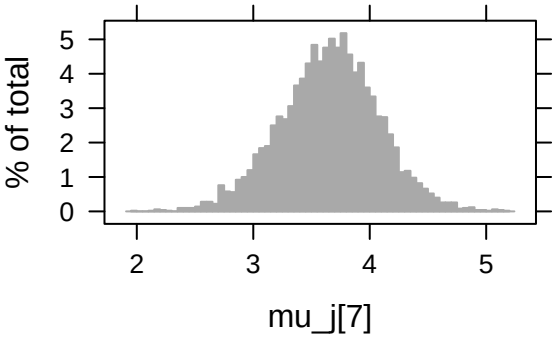

ECDF
0.0 0.2 0.4 0.6 0.8 1.0
2 3 4 5 6 7 8
mu_j[6]

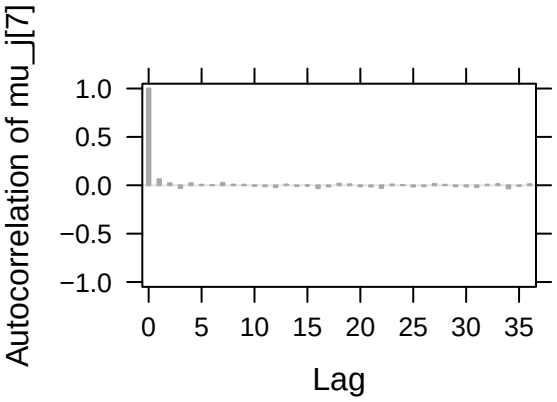

% of total
0 1 2 3
2 3 4 5 6 7 8
mu_j[6]

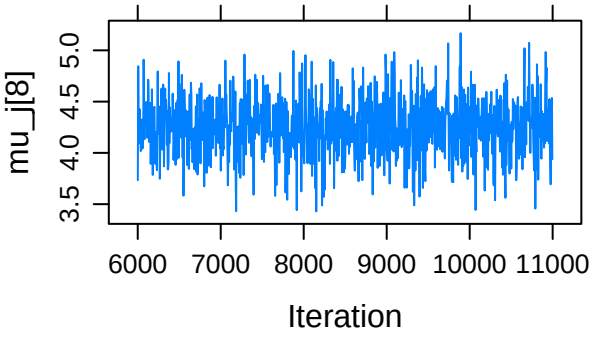

Autocorrelation of mu_j[6]
−1.0 −0.5 0.0 0.5 1.0
0 5 10 15 20 25 30 35
Lag

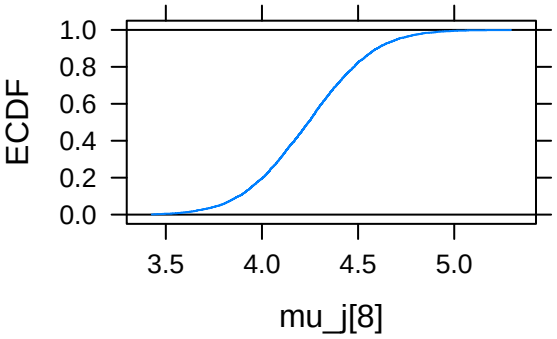

mu_j[7]
2.0
3.0
4.0
5.0
6000
7000
8000
9000
10000
11000
Iteration

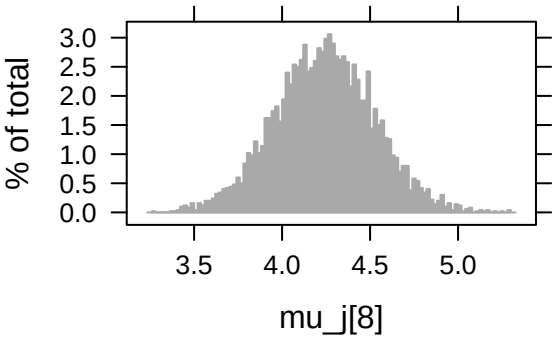

ECDF
0.0
0.2
0.4
0.6
0.8
1.0
2.5
3.0
3.5
4.0
4.5
5.0
mu_j[7]

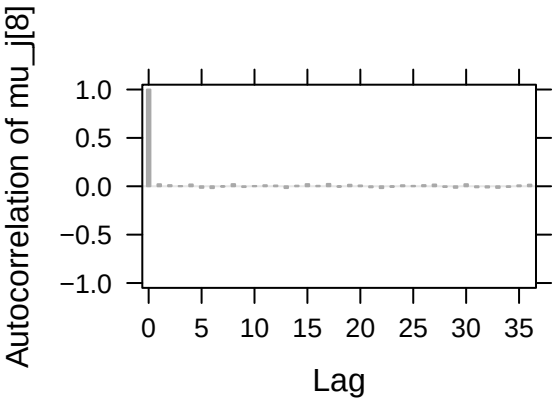

% of total
0
1
2
3
4
5
2
3
4
5
mu_j[7]

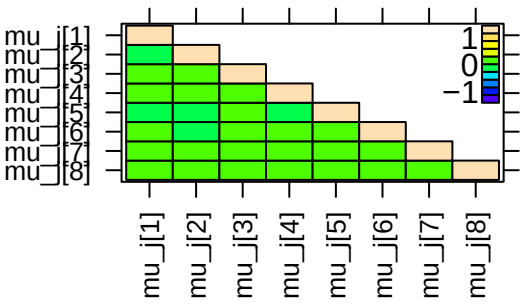

Autocorrelation of mu_j[7]
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
25
30
35
Lag


mu_j[8]
3.5
4.0
4.5
5.0
6000
7000
8000
9000
10000
11000
Iteration


ECDF
0.0
0.2
0.4
0.6
0.8
1.0
3.5
4.0
4.5
5.0
mu_j[8]

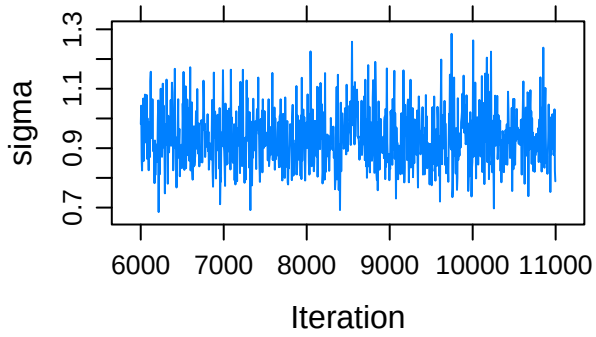

% of total
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
mu_j[8]

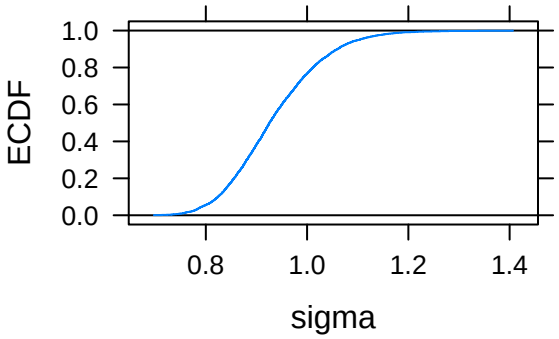

Autocorrelation of mu_j[8]
−1.0
−0.5
0.0
0.5
1.0
0
5
10
15
20
25
30
35
Lag

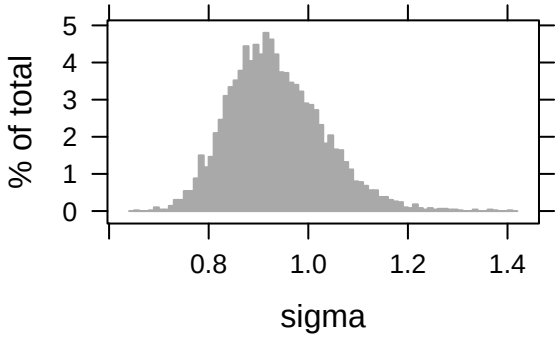


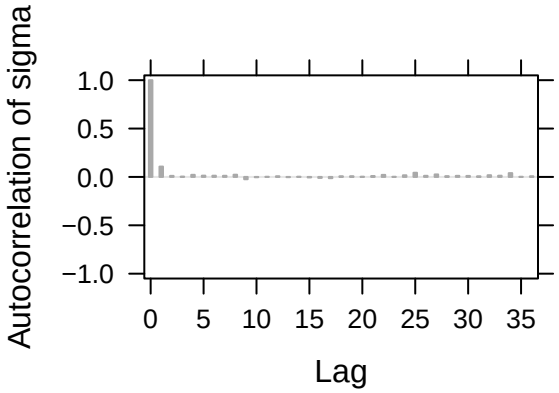

```
plot(posterior, vars = "sigma")
```

```
## Generating plots...
```









Mostly the samples seem to have converged, judging from trace and autocorrelation plots. Only mu_2 seems to have some amount of stickiness while *sigma* has a little bit of correlation but neither are sizeable enough to necessitate higher samlpes, thinning, or burn-in.

c) Shrinkage/Pooling

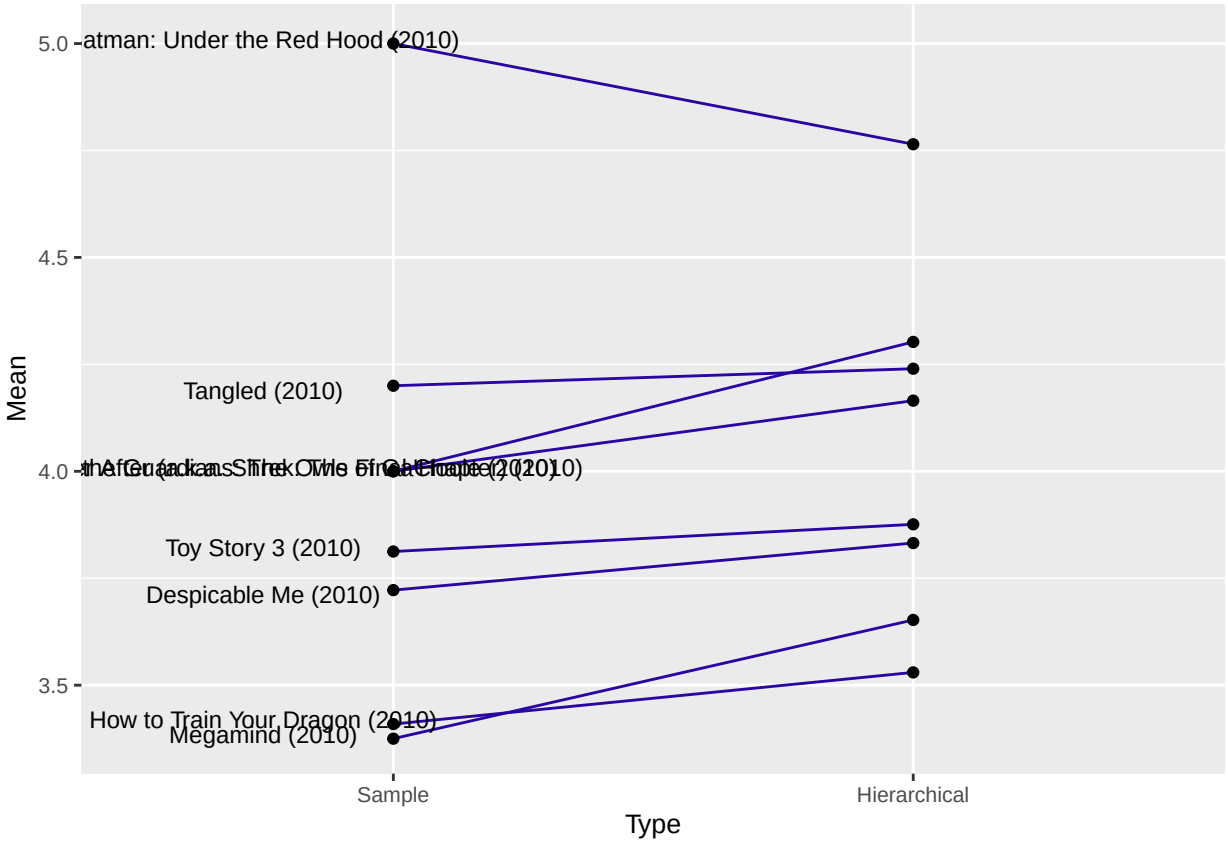
Shrinkage: We compare the shrinkage effects in the mean ratings from the sample and those given by the posterior.

```
MovieTitles <- X2010_animation_ratings %>% arrange(Group_Number) %>% select(title) %>% u
```

```
Ind_Stats = as.data.frame(matrix(NA, J, 2))
names(Ind_Stats) = c("mean", "sd")
for (j in 1:J){
```

```
  Ind_Stats[j, ] = c(mean(X2010_animation_ratings$rating[X2010_animation_ratings$Group_N
}

Post_Means <- summary(posterior)[, 4]

Means1 <- data.frame(Type = "Sample", Mean = Ind_Stats$mean)
Means2 <- data.frame(Type = "Hierarchical", Mean =
                       Post_Means[3:(4 + J - 2)])


Means1$Title <- MovieTitles$title
Means2$Title <- MovieTitles$title
```

```
ggplot(rbind(Means1, Means2), aes(Type, Mean, group=Title)) +
  geom_line(color = crcblue) + geom_point() +
  annotate(geom = "text", x = 0.75,
           y = Means1$Mean + c(0.01, 0.01, 0.01, -0.01),
           size = 3, label = Means1$Title) + increasefont(Size = 10)
```



A large pooling effect is thus seen in the posterior means. This is also reflected through low variability.

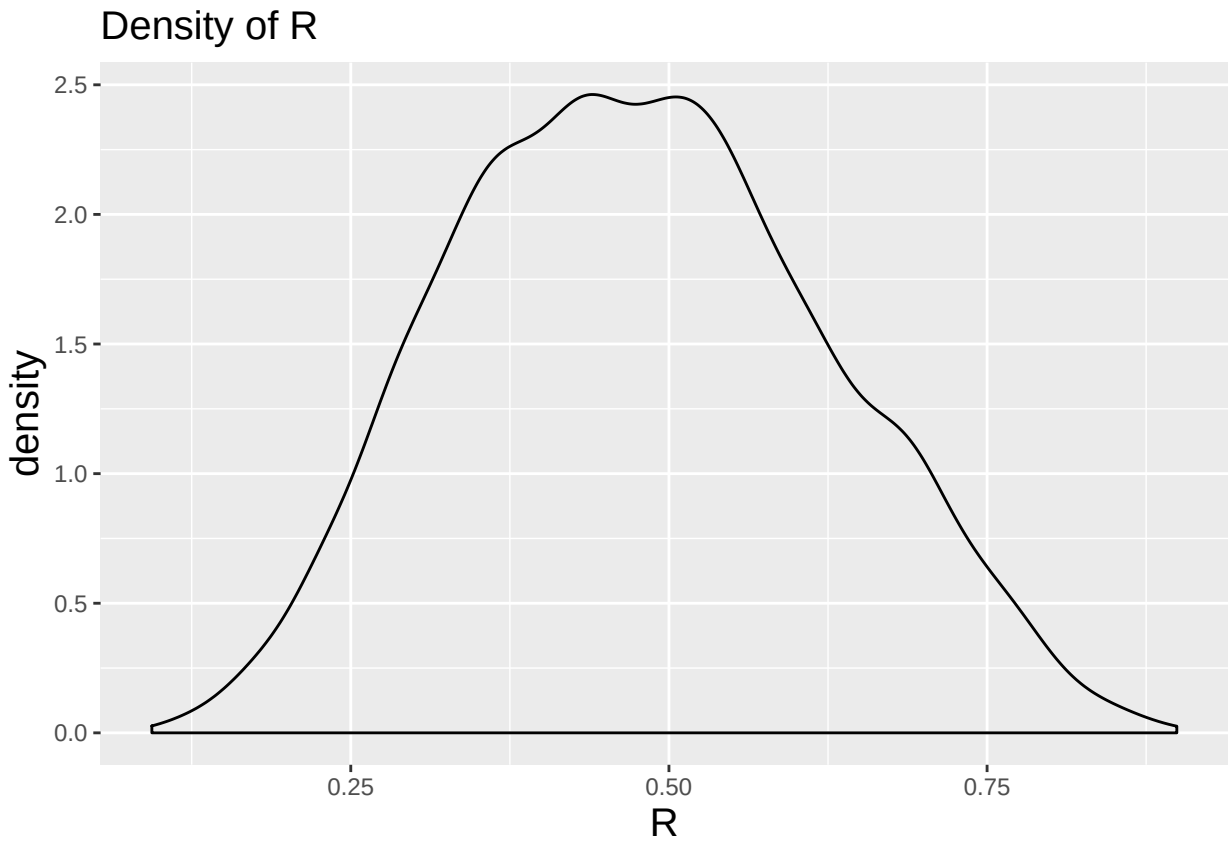
Sources of variability: As with Normal hierarchical models, the in-group variability comes

from σ and the between-group variability comes from τ. We look at the parameter $R = \frac{\tau^2}{\sigma^2+\tau^2}$:

```
require(coda)
tau_draws <- as.mcmc(posterior, vars = "tau")
sigma_draws <- as.mcmc(posterior, vars = "sigma")
R <- tau_draws^2/(tau_draws^2 + sigma_draws^2)

df <- as.data.frame(R)

quantile(R, c(0.025, 0.975))
```

```
##      2.5%     97.5%
## 0.2162531 0.7679365
```

```
ggplot(df, aes(x=R)) + geom_density() +
  labs(title="Density of R") +
  theme(plot.title = element_text(size=15)) +
  theme(axis.title = element_text(size=15))
```



R is closer to 0.5 so the inter-group variability is low.

Problem 2

a) Let $Y(x)$ denote the price of a house of size x. Then I use a linear regression model with a weakly informative prior:

$$\begin{aligned} Y(x) &\sim N(\beta_0 + \beta_1(x), \sigma) \\ \beta_0 &\sim N(\mu_0, s_0) \\ \beta_1 &\sim N(\mu_1, s_1) \\ 1/\sigma^2 &\sim \text{Gamma}(a, b) \end{aligned}$$

b) Using JAGS:

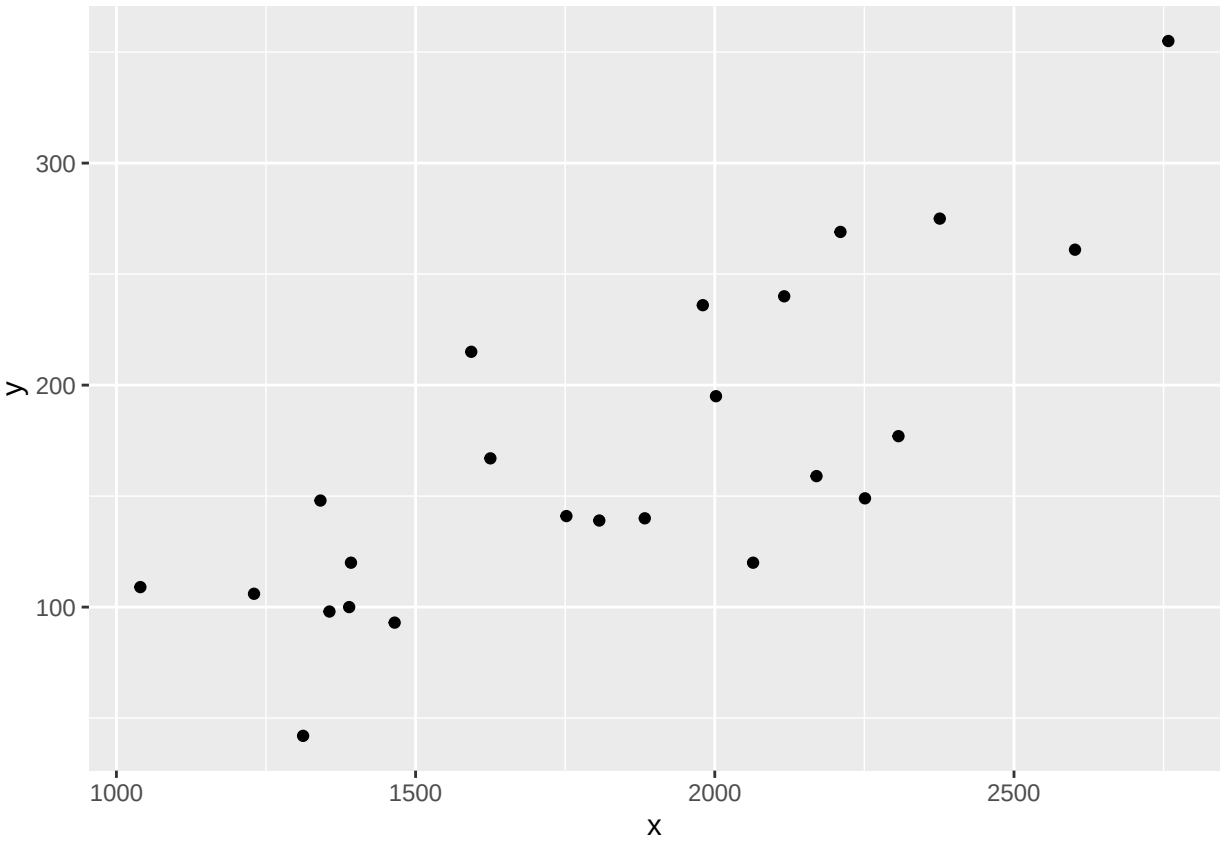
```
library(readr)
house_prices <- read_csv("house_prices.csv")
```

```
## Parsed with column specification:
## cols(
##   price = col_double(),
##   size = col_double()
## )
```

```
#View(house_prices)
y <- house_prices$price
x <- house_prices$size
N <-length(y)
```

```
ggplot(as.data.frame(x,y), aes(x=x, y=y))+
   geom_point()
```

```
## Warning in as.data.frame.numeric(x, y): 'row.names' is not a character
## vector of length 24 -- omitting it. Will be an error!
```




```
modelString <-"
model {
## sampling
for (i in 1:N){
y[i] ~ dnorm(beta0 + beta1*x[i], invsigma2)
}

## priors
beta0 ~ dnorm(mu0, g0)
beta1 ~ dnorm(mu1, g1)
invsigma2 ~ dgamma(a, b)
sigma <- sqrt(pow(invsigma2, -1))
}
"
```

```
the_data <- list("y" = y, "x" = x, "N" = N,
                 "mu0" = 0, "g0" = 0.0001,
                 "mu1" = 0, "g1" = 0.0001,
                 "a" = 1, "b" = 1)

initsfunction <- function(chain){
  .RNG.seed <- c(1,2)[chain]
  .RNG.name <- c("base::Super-Duper",
```

```
                  "base::Wichmann-Hill")[chain]
  return(list(.RNG.seed=.RNG.seed,
              .RNG.name=.RNG.name))
}
```

```
posterior <- run.jags(modelString,
                      n.chains = 1,
                      data = the_data,
                      monitor = c("beta0", "beta1", "sigma"),
                      adapt = 1000,
                      burnin = 5000,
                      sample = 10000,
                      thin = 50,
                      inits = initsfunction)
```

```
## Calling the simulation...
## Welcome to JAGS 4.3.0 on Sun Nov 24 22:51:49 2019
## JAGS is free software and comes with ABSOLUTELY NO WARRANTY
## Loading module: basemod: ok
## Loading module: bugs: ok
## . . Reading data file data.txt
## . Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
## Graph information:
##    Observed stochastic nodes: 24
##    Unobserved stochastic nodes: 3
##    Total graph size: 110
## . Reading parameter file inits1.txt
## . Initializing model
## . Adaptation skipped: model is not in adaptive mode.
## . Updating 5000
## -------------------------------------------------| 5000
## ************************************************** 100%
## . . . . Updating 500000
## -------------------------------------------------| 500000
## ************************************************** 100%
## . . . . Updating 0
## . Deleting model
## .
## Note: the model did not require adaptation
## Simulation complete.  Reading coda files...
## Coda files loaded successfully
## Calculating summary statistics...
```

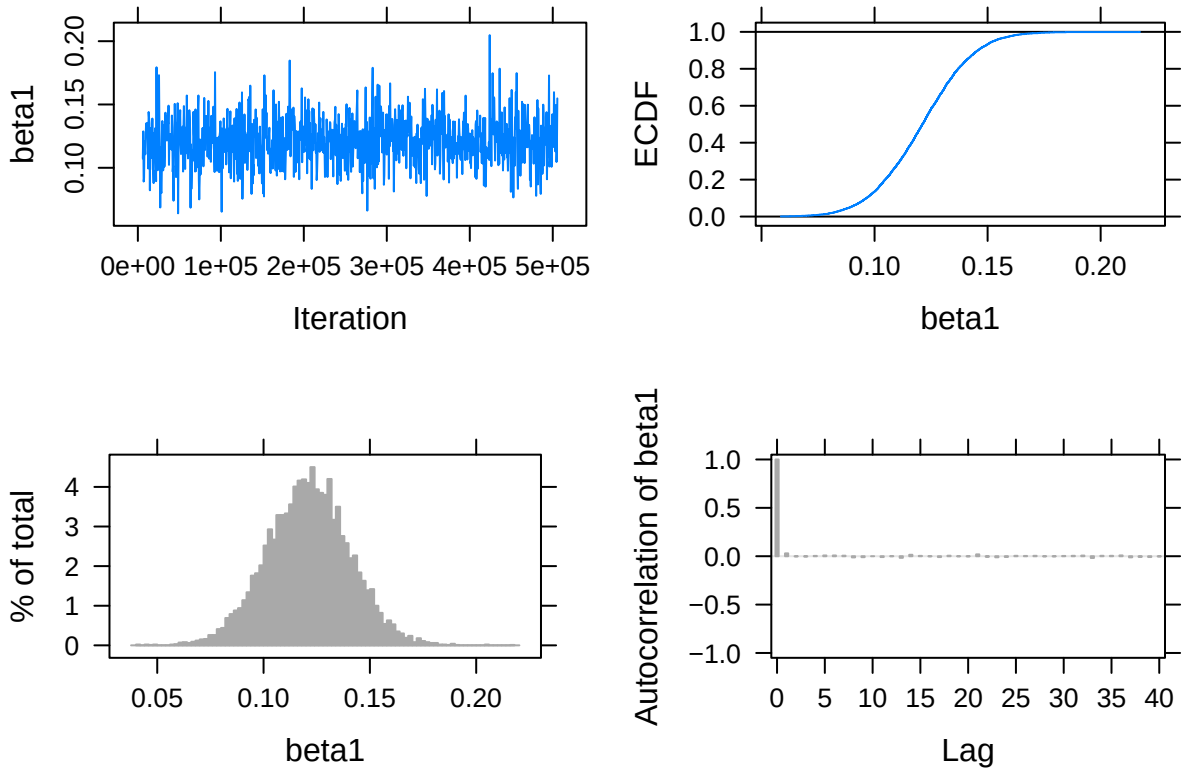
```
## Warning: Convergence cannot be assessed with only 1 chain

## Finished running the simulation
```

```
summary(posterior)
```

```
##            Lower95     Median  Upper95        Mean          SD Mode
## beta0 -125.1760000 -53.051650 18.59690 -52.7509355 36.50424739   NA
## beta1    0.0821146   0.121189  0.15854   0.1210461  0.01947499   NA
## sigma   33.1135000  44.573450 59.47570  45.4681989  6.94985481   NA
##               MCerr MC%ofSD SSeff         AC.500 psrf
## beta0 0.3743745123       1  9508 -2.690072e-03   NA
## beta1 0.0002005912       1  9426  2.671387e-05   NA
## sigma 0.0694985481       1 10000  8.108257e-04   NA
```

```
plot(posterior,  vars = "beta1")
```

```
## Generating plots...
```



Here I have plotted the MCMC diagnostics for the β_1, i.e the slope parameter. MCMC convergence can be detected through the relatively spread-out trace plot and rapidly decaying autocorrelation (I used 10000 samplels with a thinning rate of 50).

c) The intercept β_0: The results for β_0 indicate that an apartment of size 0 has , on average, a price of -52.75K USD and its price falls in $(-125.18\text{K}, 18.60\text{K})$ 90% of the time.

The slope β_1: When the size of a unit goes up by 1000 sq.ft, its price, on average, goes up by USD 0.12K and 90% of the time the rise in price is between USD `0.0821146`K and `0.15854`K.

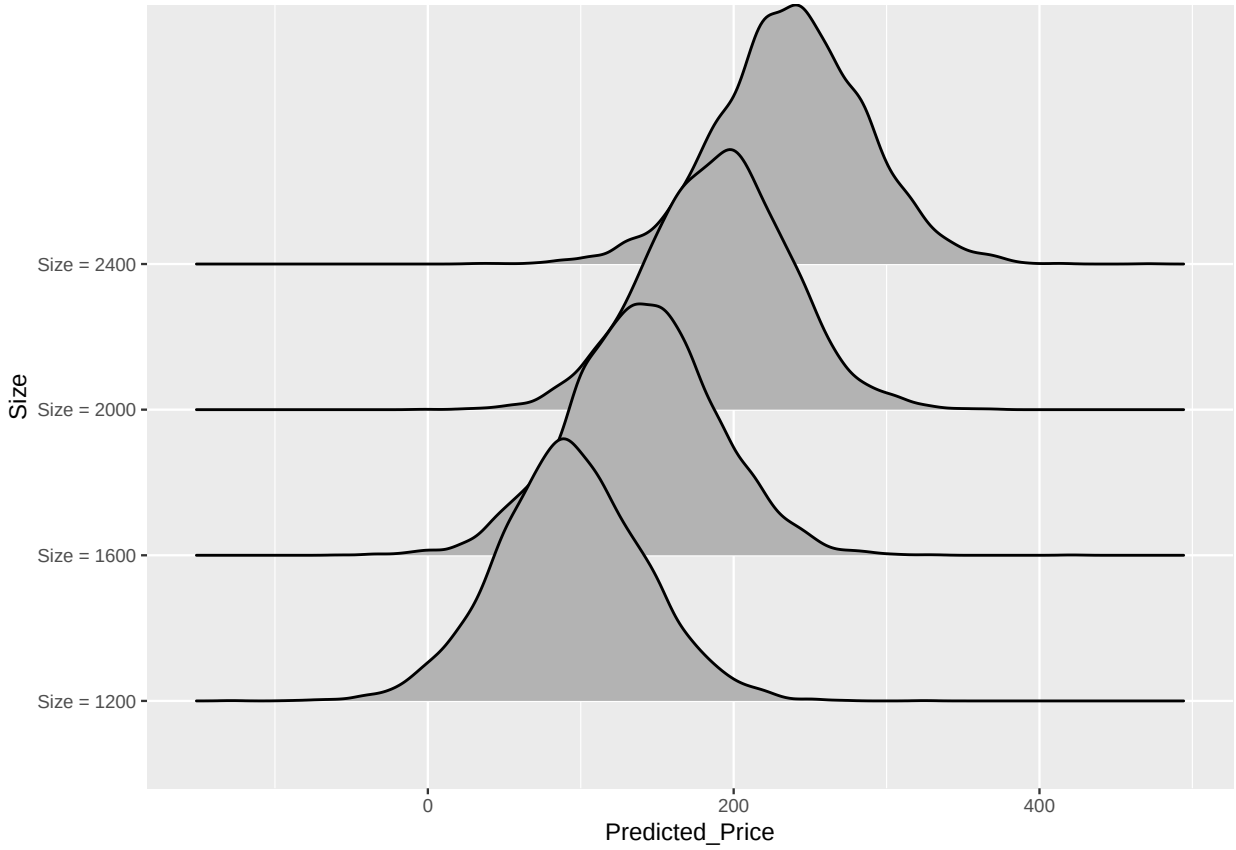
d)

Note that the expected price of an apartment is a good estimator of its predicted price and $E[Y(x)] = E[\beta_0] + xE[\beta_1]$.

```
##   Sizes Exp_Prices
## 1  1200   92.50438
## 2  1600  140.92282
## 3  2000  189.34126
## 4  2400  237.75970
```

Finally, the 90% credible intervals can be visualized and summarized as follows:

```
require(ggridges)
ggplot(df, aes(x = Predicted_Price, y = Size)) +
  geom_density_ridges() +
  theme_grey(base_size = 9, base_family = "")
```




```
df %>% group_by(Size) %>%
  summarize(P05 = quantile(Predicted_Price, 0.05),
            P95 = quantile(Predicted_Price, 0.95))
```

```
## # A tibble: 4 x 3
##   Size          P05   P95
##   <chr>       <dbl> <dbl>
## 1 Size = 1200  15.3  173.
## 2 Size = 1600  59.5  219.
## 3 Size = 2000 111.   264.
## 4 Size = 2400 160.   316.
```