analysis_2-10

Kevin Ros

2/6/2020

Collaboration: I wasn't sure where to start, and Henrik pointed me in the direction of last year's multiple linear regression slides: https://github.com/monika76five/Undergrad-Bayesian-Statistics/blob/master/Lectures/BayesianLinearRegression/F19MATH347_Regression_R.pdf which I implemented, as it wasn't too clear how to do hierarchical modeling with categorical variables (but I think fastDummies can be used). Additionally, Henrik and I discussed the simulation portion, namely how one needs to pass a data frame of each variable, and not just income.

```r
library(ggplot2)
library(coda)
library(runjags)
library(fastDummies)
data = data.frame(read.csv("../datasets/CEdata.csv",header=TRUE))
```

```r
# EXAMPLE FROM CLASS

# 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))
# }
# "
#
# y <- as.vector(log(data$TotalIncomeLastYear))
# x <- as.vector(log(data$TotalExpLastQ))
# N <-length(y)
# 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::SuperDuper",
#                  "base::Wichmann-Hill")[chain]
#   return(list(.RNG.seed=.RNG.seed,
#               .RNG.name-.RNG.name))
# }
# ```
#
# ```{r}
```

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

The original slides has scale(), but I removed it to better interpret the results.

```
data$log_TotalExpSTD <- log(data$Expenditure)
data$log_TotalIncomeSTD <- log(data$Income)
```

Create the binary columns and rows for each categorical variable:

```
data$Rural = fastDummies::dummy_cols(data$UrbanRural)[,names(fastDummies::dummy_cols(data$UrbanRural))==
data$Race_Black = fastDummies::dummy_cols(data$Race)[,names(fastDummies::dummy_cols(data$Race)) == ".da
data$Race_NA = fastDummies::dummy_cols(data$Race)[,names(fastDummies::dummy_cols(data$Race)) == ".data_
data$Race_Asian = fastDummies::dummy_cols(data$Race)[,names(fastDummies::dummy_cols(data$Race)) == ".da
data$Race_PI = fastDummies::dummy_cols(data$Race)[,names(fastDummies::dummy_cols(data$Race)) == ".data_
data$Race_M = fastDummies::dummy_cols(data$Race)[,names(fastDummies::dummy_cols(data$Race)) == ".data_6
```

Same parameters from slides:

```
modelString <-"
model {
## sampling
for (i in 1:N){
y[i] ~ dnorm(beta0 + beta1*x_income[i] + beta2*x_rural[i] +
beta3*x_race_B[i] + beta4*x_race_N[i] +
beta5*x_race_A[i] + beta6*x_race_P[i] +
beta7*x_race_M[i], invsigma2)
}
## priors
beta0 ~ dnorm(mu0, g0)
beta1 ~ dnorm(mu1, g1)
beta2 ~ dnorm(mu2, g2)
beta3 ~ dnorm(mu3, g3)
beta4 ~ dnorm(mu4, g4)
beta5 ~ dnorm(mu5, g5)
beta6 ~ dnorm(mu6, g6)
beta7 ~ dnorm(mu7, g7)
invsigma2 ~ dgamma(a, b)
sigma <- sqrt(pow(invsigma2, -1))
}
"
```

```
y = as.vector(data$log_TotalExpSTD)
x_income = as.vector(data$log_TotalIncomeSTD)
x_rural = as.vector(data$Rural)
x_race_B = as.vector(data$Race_Black)
x_race_N = as.vector(data$Race_NA)
x_race_A = as.vector(data$Race_Asian)
x_race_P = as.vector(data$Race_PI)
```

```
x_race_M = as.vector(data$Race_M)
N = length(y)
```

Same parameter values from slides:

```
the_data <- list("y" = y, "x_income" = x_income,
                 "x_rural" = x_rural, "x_race_B" = x_race_B,
                 "x_race_N" = x_race_N, "x_race_A" = x_race_A,
                 "x_race_P" = x_race_P, "x_race_M" = x_race_M,
                 "N" = N,
                 "mu0" = 0, "g0" = 1, "mu1" = 0, "g1" = 1,
                 "mu2" = 0, "g2" = 1, "mu3" = 0, "g3" = 1,
                 "mu4" = 0, "g4" = 1, "mu5" = 0, "g5" = 1,
                 "mu6" = 0, "g6" = 1, "mu7" = 0, "g7" = 1,
                 "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))
}
```

Thinning of 50 is needed, otherwise beta0 and beta1 have very high lag

```
posterior_MLR <- run.jags(modelString,
                      n.chains = 1,
                      data = the_data,
                      monitor = c("beta0", "beta1", "beta2",
                                  "beta3", "beta4", "beta5",
                                  "beta6", "beta7", "sigma"),
                      adapt = 1000,
                      burnin = 5000,
                      sample = 5000,
                      thin = 50,
                      inits = initsfunction)
```

```
## Loading required namespace: rjags

## Compiling rjags model...
## Calling the simulation using the rjags method...
## Note: the model did not require adaptation
## Burning in the model for 5000 iterations...
## Running the model for 250000 iterations...
## Simulation complete
## Calculating summary statistics...

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

## Finished running the simulation
```

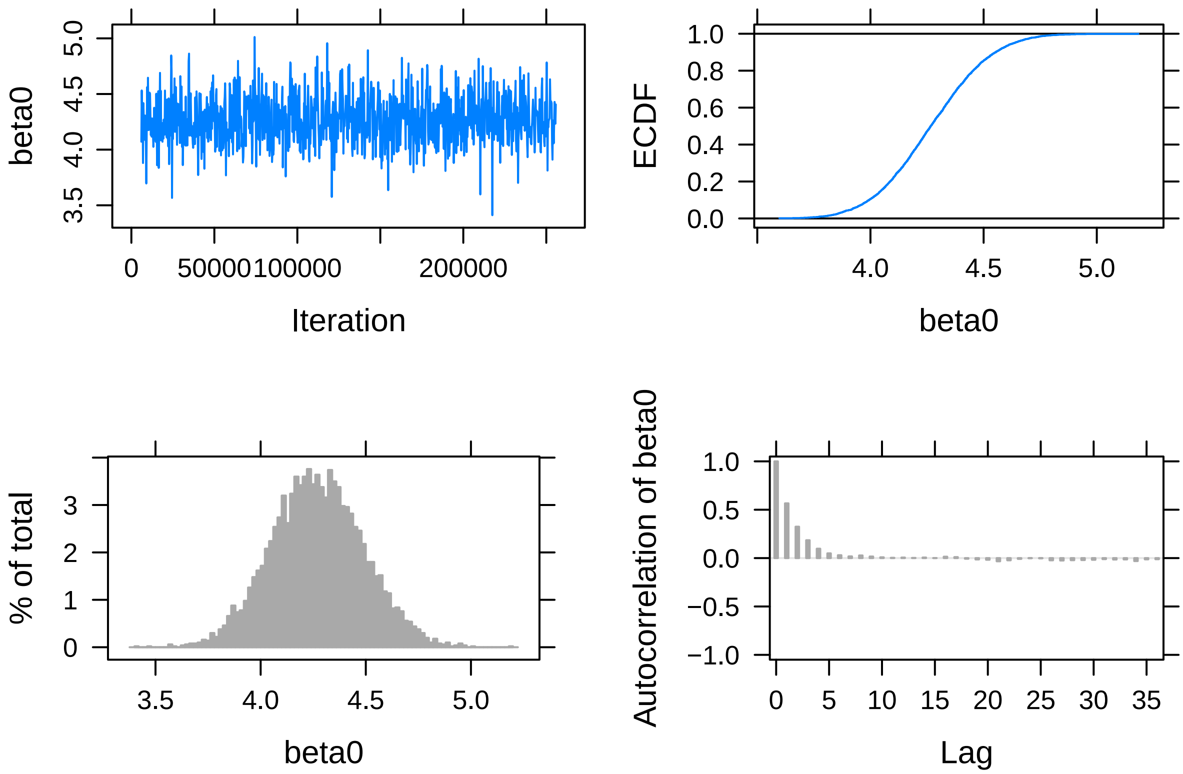
```
summary(posterior_MLR)
```

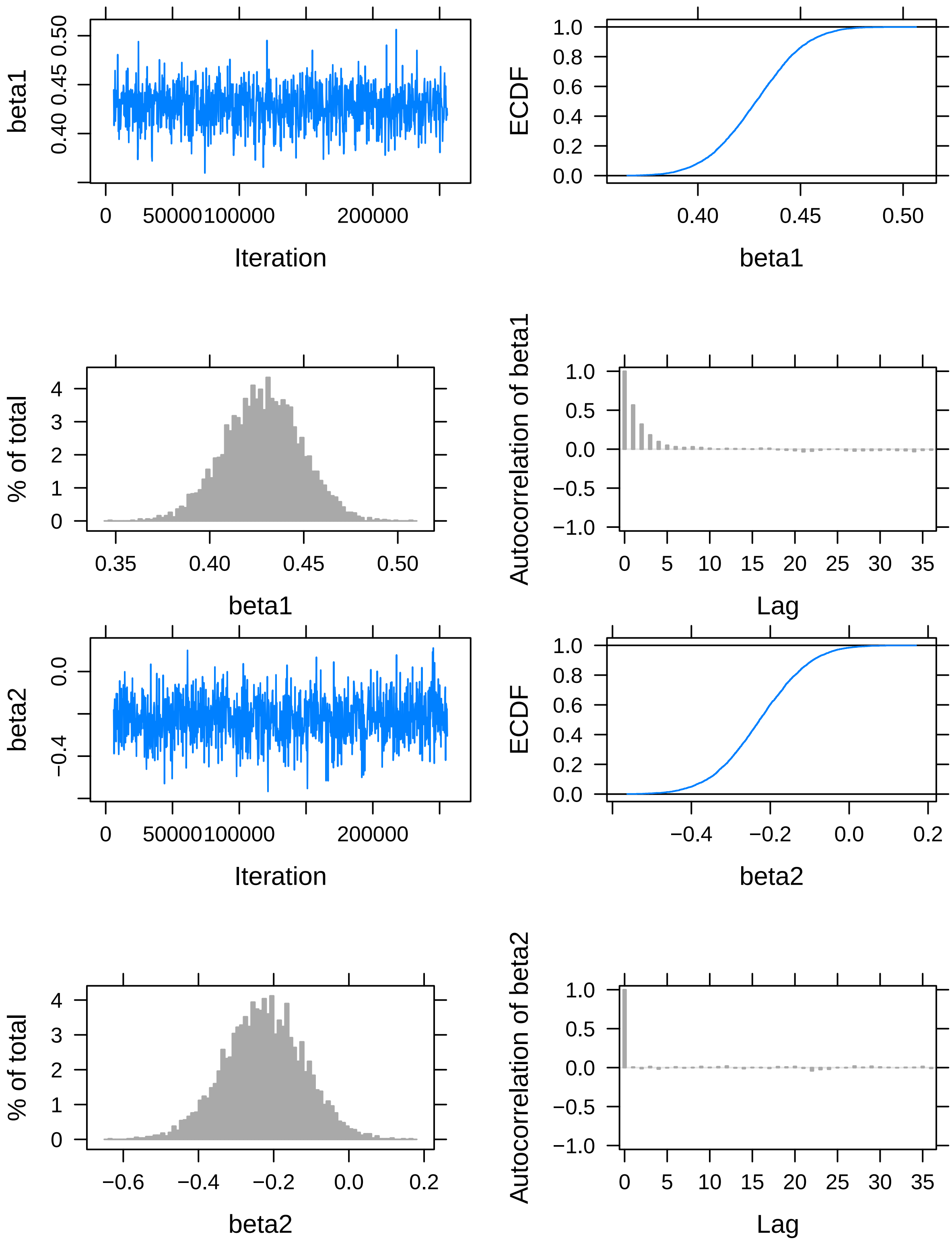
```
##          Lower95       Median     Upper95         Mean         SD Mode
## beta0  3.8401288  4.265856654  4.68625457  4.270064662 0.21777079   NA
## beta1  0.3882419  0.428382748  0.46702186  0.428186927 0.02024773   NA
## beta2 -0.4324816 -0.227295748 -0.02303789 -0.226517253 0.10523080   NA
```

```
## beta3 -0.3306556 -0.190482478 -0.04140575 -0.191212993 0.07415444    NA
## beta4 -0.5206527  0.004840372  0.51719407  0.004623109 0.26616225    NA
## beta5 -0.0510838  0.162814921  0.40645797  0.162083182 0.11830143    NA
## beta6 -0.4595443  0.083013003  0.68580148  0.082703016 0.28996472    NA
## beta7 -0.2892273  0.040916578  0.39024470  0.042723529 0.17484430    NA
## sigma  0.6904107  0.721247329  0.75457051  0.721355983 0.01634697    NA
##              MCerr MC%ofSD SSeff        AC.500 psrf
## beta0 0.0058531144     2.7  1384  8.671215e-03   NA
## beta1 0.0005445194     2.7  1383  1.118156e-02   NA
## beta2 0.0014881883     1.4  5000  3.940920e-03   NA
## beta3 0.0010321668     1.4  5161  1.701377e-03   NA
## beta4 0.0037641027     1.4  5000  6.253260e-03   NA
## beta5 0.0017356791     1.5  4646  3.048365e-02   NA
## beta6 0.0041007204     1.4  5000  9.247818e-03   NA
## beta7 0.0024726717     1.4  5000  1.516543e-02   NA
## sigma 0.0002311811     1.4  5000 -1.238722e-05   NA
```

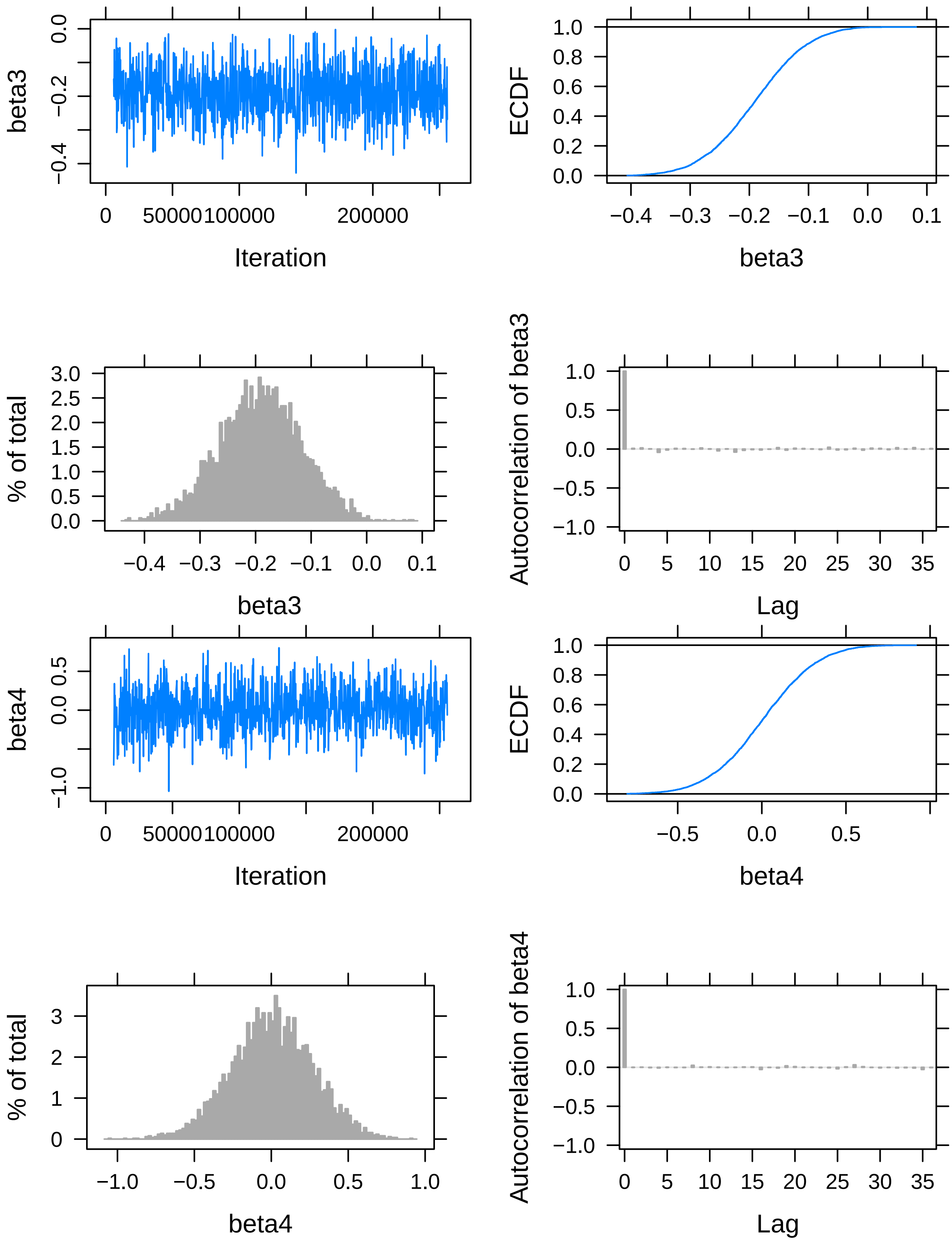
```
plot(posterior_MLR)
```

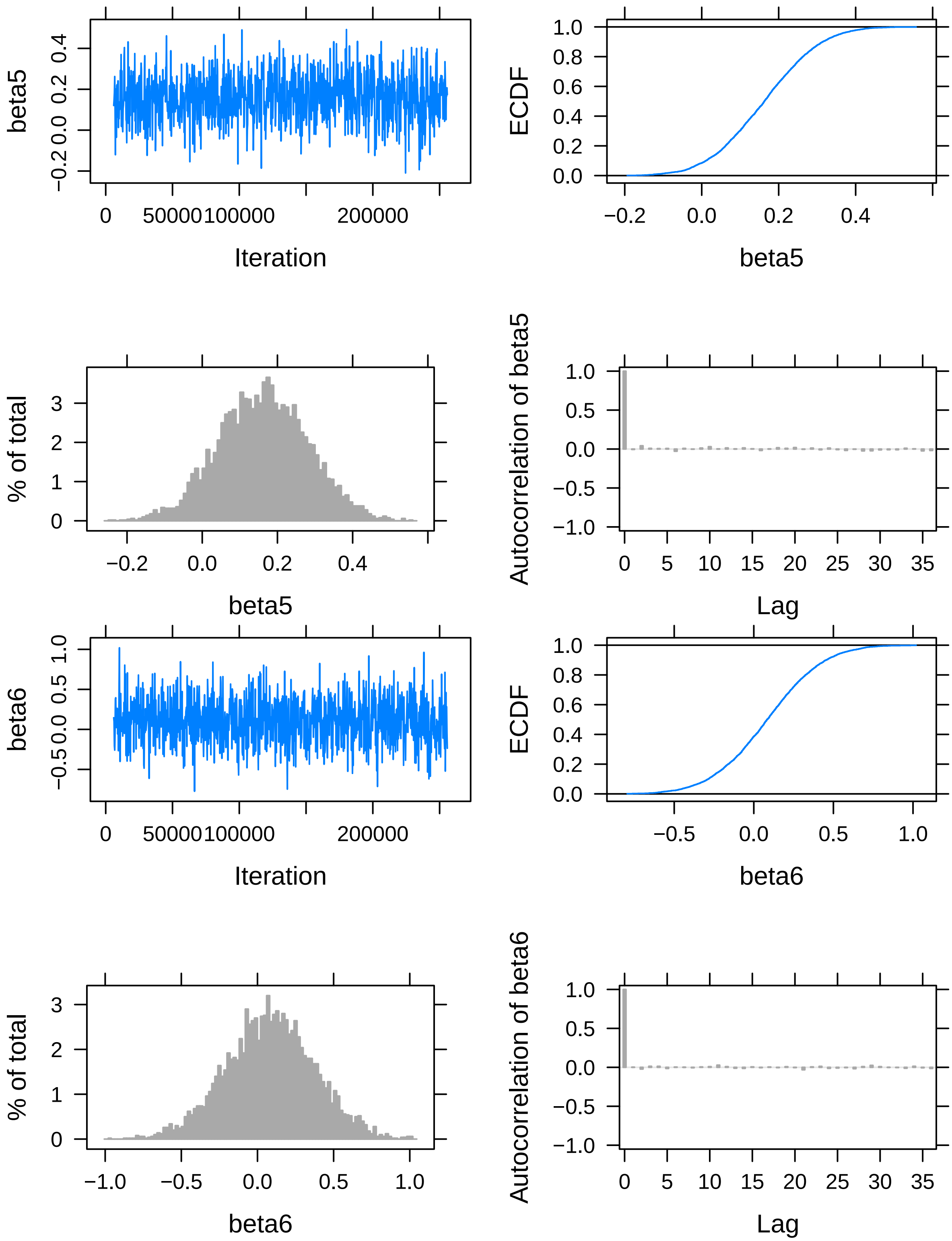
```
## Generating plots...
```

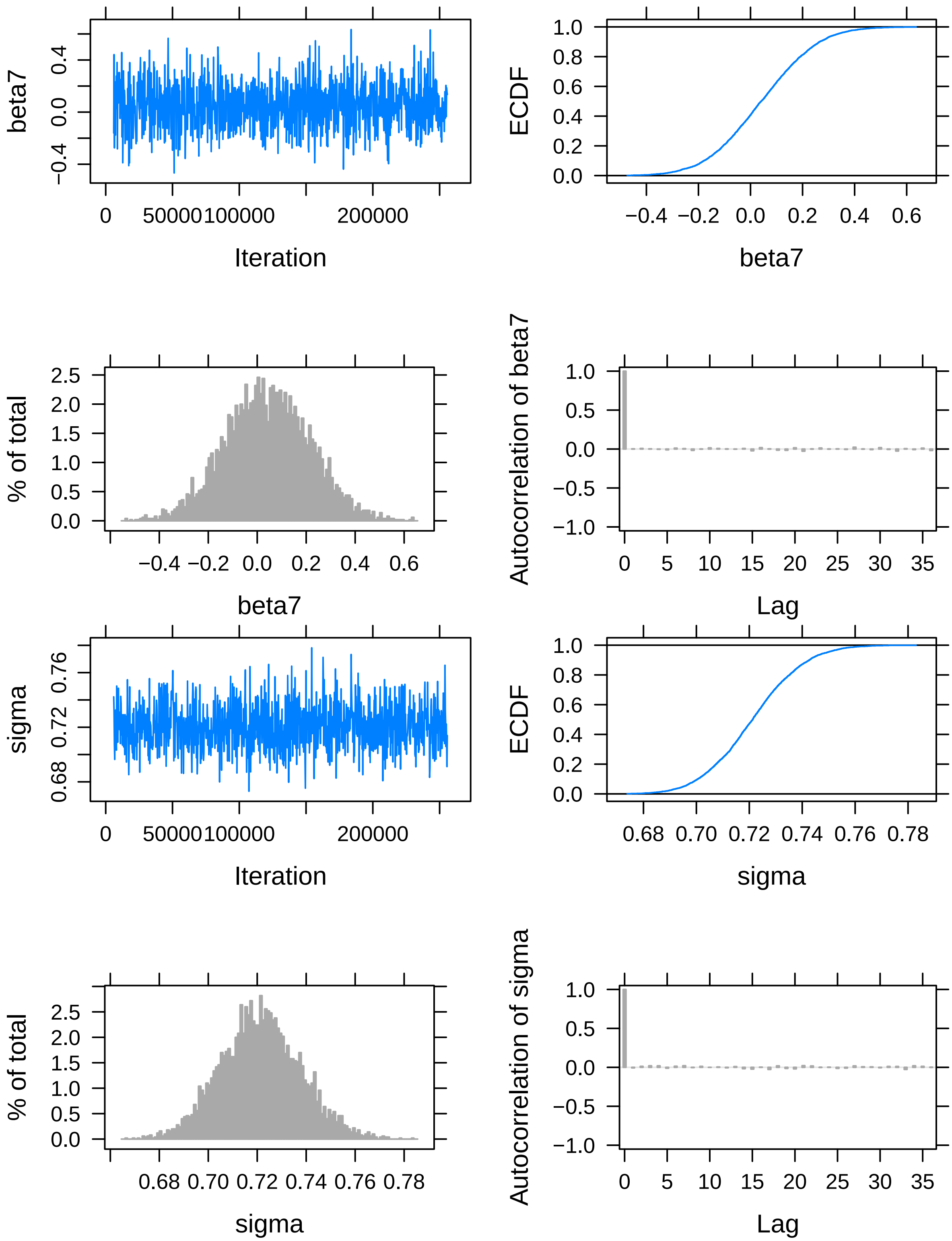


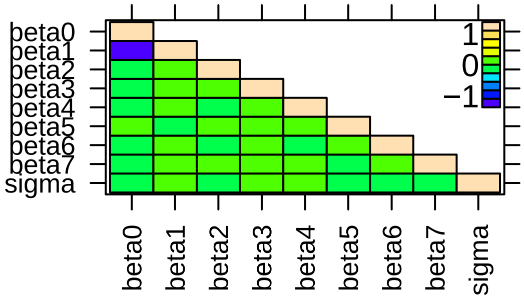


beta1
0.40 0.45 0.50
0 50000100000 200000
Iteration
ECDF
1.0 0.8 0.6 0.4 0.2 0.0
0.40 0.45 0.50
beta1
% of total
4 3 2 1 0
0.35 0.40 0.45 0.50
beta1
Autocorrelation of beta1
1.0 0.5 0.0 −0.5 −1.0
0 5 10 15 20 25 30 35
Lag
beta2
0.0 −0.4
0 50000100000 200000
Iteration
ECDF
1.0 0.8 0.6 0.4 0.2 0.0
−0.4 −0.2 0.0 0.2
beta2
% of total
4 3 2 1 0
−0.6 −0.4 −0.2 0.0 0.2
beta2
Autocorrelation of beta2
1.0 0.5 0.0 −0.5 −1.0
0 5 10 15 20 25 30 35
Lag


beta3
Iteration
0
50000100000
200000
0.0
−0.2
−0.4
ECDF
beta3
−0.4
−0.3
−0.2
−0.1
0.0
0.1
% of total
3.0
2.5
2.0
1.5
1.0
0.5
Autocorrelation of beta3
Lag
0
5
10
15
20
25
30
35
1.0
−0.5
−1.0
beta4
0.5
−1.0
−0.5
Autocorrelation of beta4


beta5
Iteration
ECDF
% of total
Autocorrelation of beta5
Lag
beta6
Autocorrelation of beta6


beta7
Iteration
ECDF
beta7
% of total
beta7
Autocorrelation of beta7
Lag
sigma
Iteration
ECDF
sigma
% of total
sigma
Autocorrelation of sigma
Lag

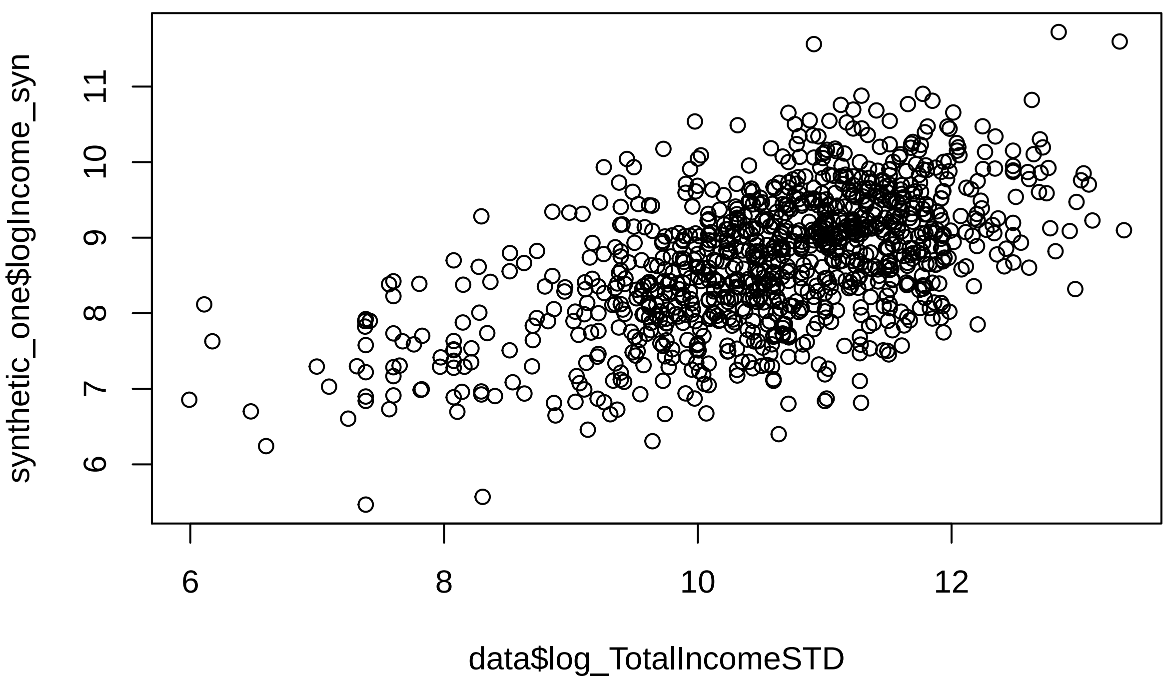



```
post <- as.mcmc(posterior_MLR)
```

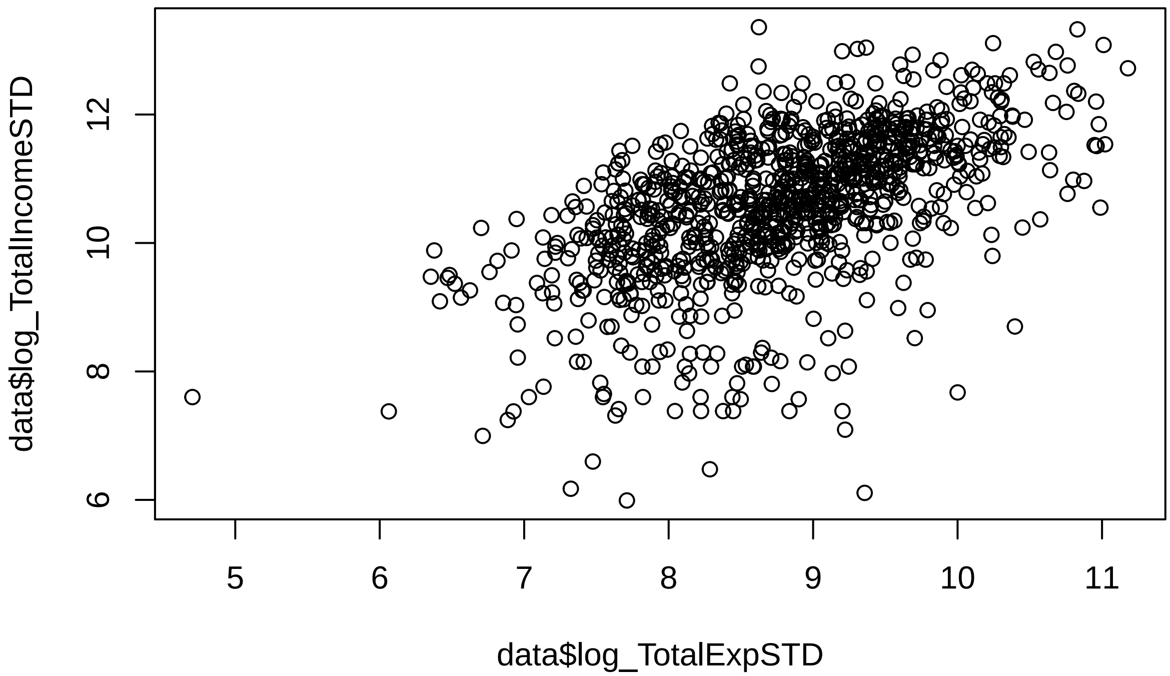
```
synthesize <- function(X, index, n){
  mean_Y <- post[index, "beta0"] + X$x_income * post[index, "beta1"] + X$x_rural * post[index, "beta2"]
  synthetic_Y <- rnorm(n,mean_Y, post[index,"sigma"])
  data.frame(X$x_income, synthetic_Y)
}

n <- dim(data)[1]
params <- data.frame(x_income, x_rural, x_race_B, x_race_N, x_race_A, x_race_P, x_race_M)
synthetic_one <- synthesize(params,1,n)
names(synthetic_one) <- c("OriginalIncome", "logIncome_syn")
```

Plotting the syntheticly generated income with the original income gives us the following:

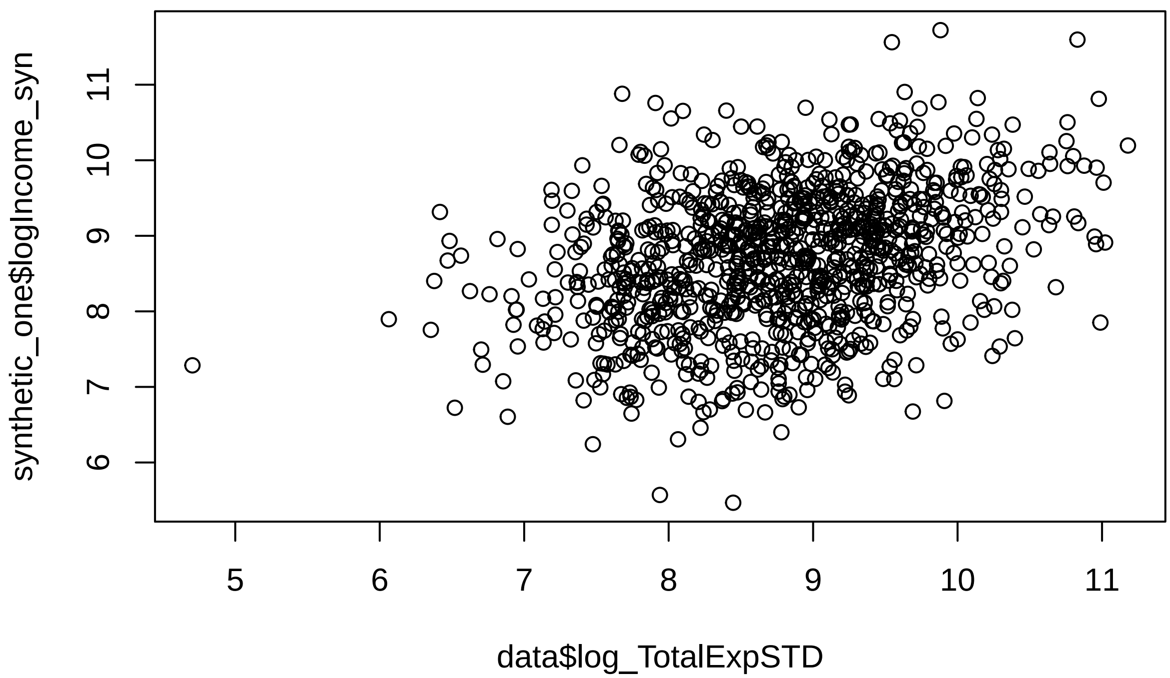
```
plot(data$log_TotalIncomeSTD, synthetic_one$logIncome_syn)
```




```
plot(data$log_TotalExpSTD, data$log_TotalIncomeSTD)
```




```r
plot(data$log_TotalExpSTD, synthetic_one$logIncome_syn)
```



The mean of the original income (10.59) is slightly more than the synthetic mean (8.77), a little more than one standard deviation.

```r
mean(synthetic_one$OriginalIncome)
```

```
## [1] 10.59507
```

```r
mean(synthetic_one$logIncome_syn)
```

```
## [1] 8.702602
```

Just as with the mean, the median of the original income (10.7) is slightly more than the median of the synthetic income median (8.81), a little more than one standard deviation.

```r
median(synthetic_one$OriginalIncome)
```

```
## [1] 10.70574
```

```r
median(synthetic_one$logIncome_syn)
```

```
## [1] 8.751067
```

```r
sd(synthetic_one$OriginalIncome)
```

```
## [1] 1.153488
```

```r
sd(synthetic_one$logIncome_syn)
```

```
## [1] 0.9120538
```

Regarding the point estimates, with an expenditure of 0, original income will be 4.3219 and synthetic income will be 5.8609. And with every unit increase of expenditure, the original income increases by 0.4211 and the synthetic income increases by 0.3333. Of course, all of these values are in log form.

```r
lm(data$log_TotalExpSTD ~ data$log_TotalIncomeSTD)
```

```
##
## Call:
## lm(formula = data$log_TotalExpSTD ~ data$log_TotalIncomeSTD)
##
## Coefficients:
##             (Intercept)  data$log_TotalIncomeSTD
##                  4.3219                   0.4211
```

```r
lm(data$log_TotalExpSTD ~ synthetic_one$logIncome_syn)
```

```
##
## Call:
## lm(formula = data$log_TotalExpSTD ~ synthetic_one$logIncome_syn)
##
## Coefficients:
##                 (Intercept)  synthetic_one$logIncome_syn
##                      6.0300                       0.3165
```