# Lesson 7: Linear regression

# Lesson 7.2

## Data

As an example of linear regression, we'll look at the Leinhardt data from the `car` package in `R`.

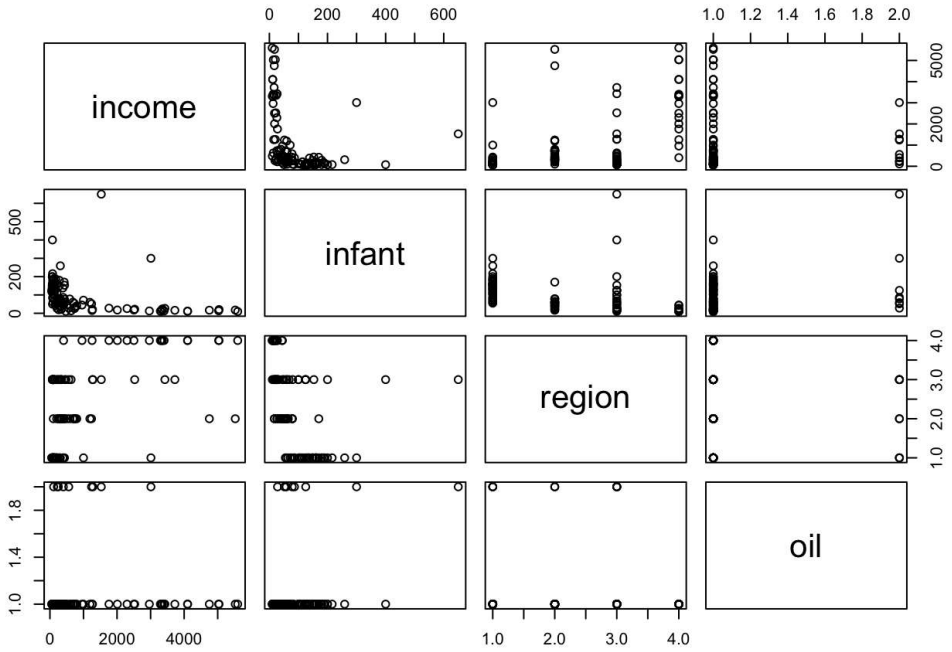
```
library("car")
data("Leinhardt")
?Leinhardt
head(Leinhardt)
```

```
##           income infant   region oil
## Australia   3426   26.7     Asia  no
## Austria     3350   23.7   Europe  no
## Belgium     3346   17.0   Europe  no
## Canada      4751   16.8 Americas  no
## Denmark     5029   13.5   Europe  no
## Finland     3312   10.1   Europe  no
```

```
str(Leinhardt)
```

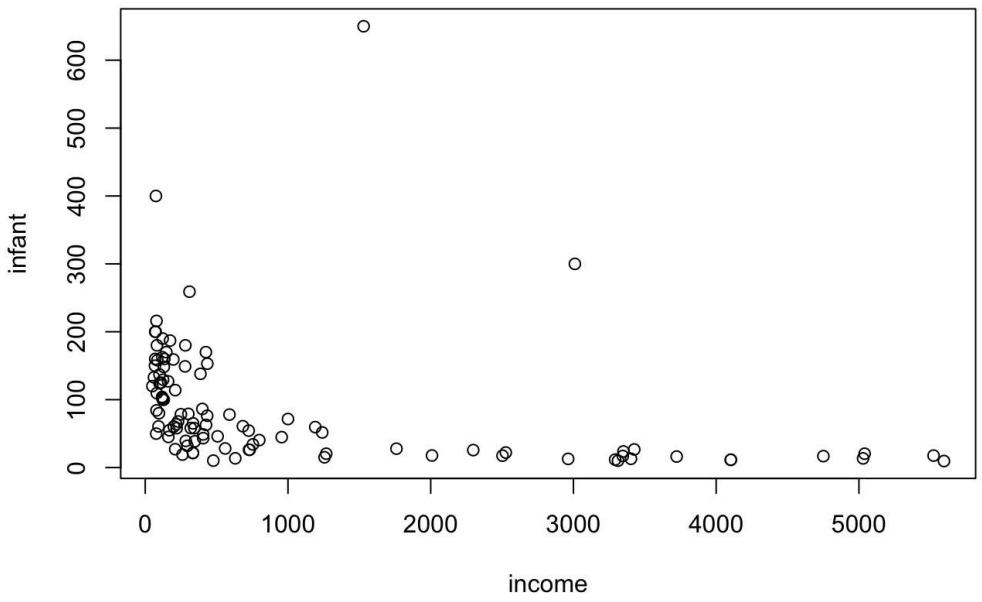
```
## 'data.frame':    105 obs. of  4 variables:
##  $ income: int  3426 3350 3346 4751 5029 3312 3403 5040 2009 2298 ...
##  $ infant: num  26.7 23.7 17 16.8 13.5 10.1 12.9 20.4 17.8 25.7 ...
##  $ region: Factor w/ 4 levels "Africa","Americas",..: 3 4 4 2 4 4 4 4 4 4 ...
##  $ oil   : Factor w/ 2 levels "no","yes": 1 1 1 1 1 1 1 1 1 1 ...
```

```
pairs(Leinhardt)
```

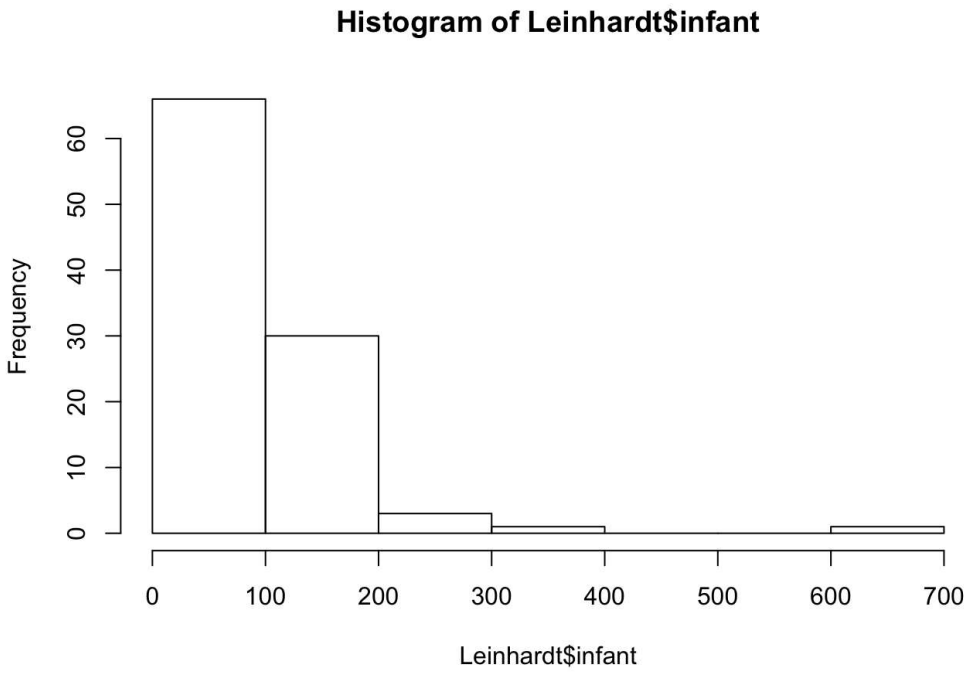



We'll start with a simple linear regression model that relates infant mortality to per capita income.

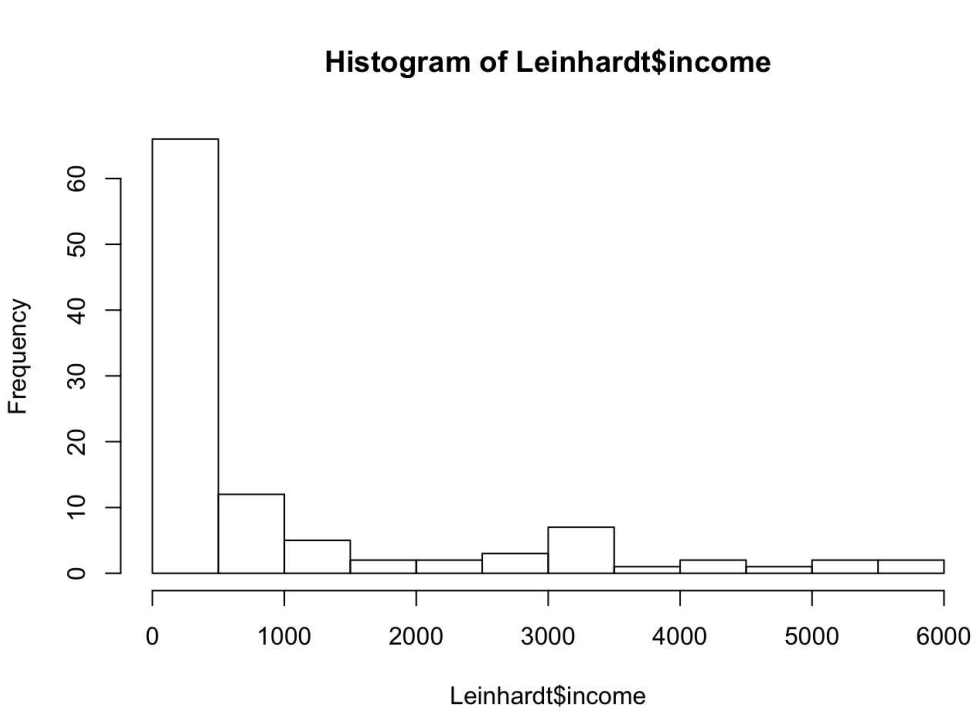
```
plot(infant ~ income, data=Leinhardt)
```



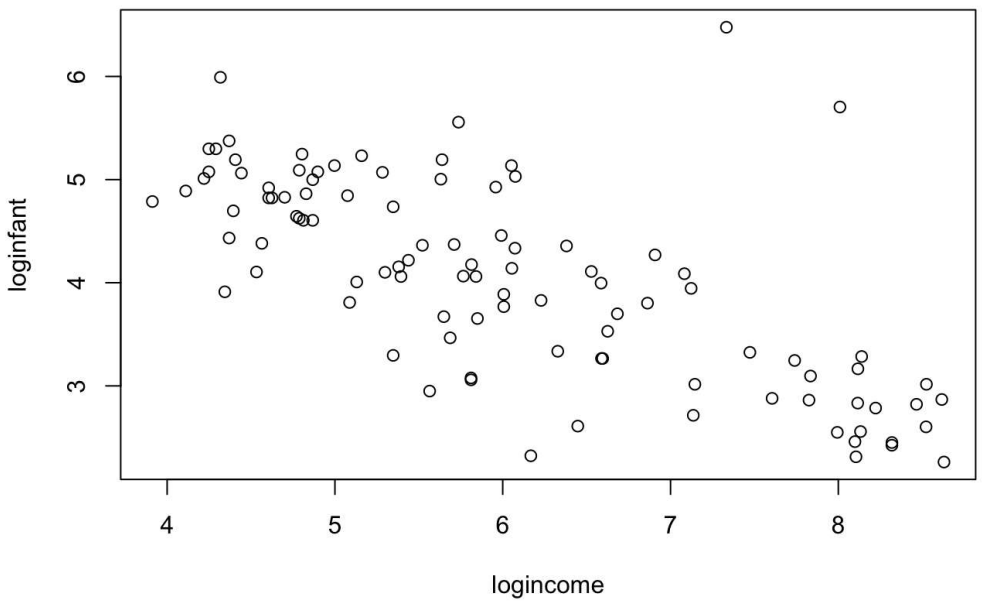

```
hist(Leinhardt$infant)
```




```
hist(Leinhardt$income)
```




```
Leinhardt$loginfant = log(Leinhardt$infant)
Leinhardt$logincome = log(Leinhardt$income)

plot(loginfant ~ logincome, data=Leinhardt)
```




Since infant mortality and per capita income are positive and right-skewed quantities, we consider modeling them on the logarithmic scale. A linear model appears much more appropriate on this scale.

# Modeling

The reference Bayesian analysis (with a noninformative prior) is available directly in `R`.

```
lmod = lm(loginfant ~ logincome, data=Leinhardt)
summary(lmod)
```

```
##
## Call:
## lm(formula = loginfant ~ logincome, data = Leinhardt)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.66694 -0.42779 -0.02649  0.30441  3.08415
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  7.14582    0.31654  22.575   <2e-16 ***
## logincome   -0.51179    0.05122  -9.992   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6867 on 99 degrees of freedom
##   (4 observations deleted due to missingness)
## Multiple R-squared:  0.5021, Adjusted R-squared:  0.4971
## F-statistic: 99.84 on 1 and 99 DF,  p-value: < 2.2e-16
```

# Leson 7.3

## Model in JAGS

Now we'll fit this model in JAGS. A few countries have missing values, and for simplicity, we will omit those.

```
dat = na.omit(Leinhardt)

library("rjags")
```

```
## Loading required package: coda
```

```
## Warning: package 'coda' was built under R version 3.3.2
```

```
## Linked to JAGS 4.2.0
```

```
## Loaded modules: basemod,bugs
```

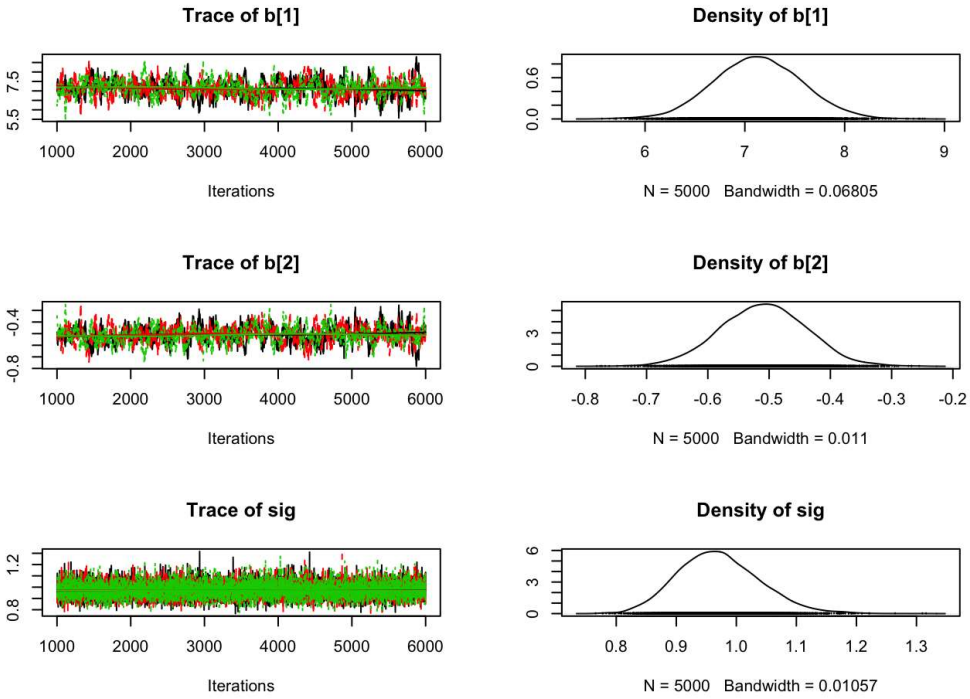
```
mod1_string = " model {
    for (i in 1:n) {
        y[i] ~ dnorm(mu[i], prec)
        mu[i] = b[1] + b[2]*log_income[i]
    }

    for (i in 1:2) {
        b[i] ~ dnorm(0.0, 1.0/1.0e6)
    }

    prec ~ dgamma(5/2.0, 5*10.0/2.0)
    sig2 = 1.0 / prec
    sig = sqrt(sig2)
} "

set.seed(72)
data1_jags = list(y=dat$loginfant, n=nrow(dat),
              log_income=dat$logincome)

params1 = c("b", "sig")

inits1 = function() {
    inits = list("b"=rnorm(2,0.0,100.0), "prec"=rgamma(1,1.0,1.0))
}

mod1 = jags.model(textConnection(mod1_string), data=data1_jags, inits=inits1, n.chains=3)
update(mod1, 1000) # burn-in

mod1_sim = coda.samples(model=mod1,
                        variable.names=params1,
                        n.iter=5000)

mod1_csim = do.call(rbind, mod1_sim) # combine multiple chains
```

# Lesson 7.4

## MCMC convergence

Before we check the inferences from the model, we should perform convergence diagnostics for our Markov chains.

```
plot(mod1_sim)
```



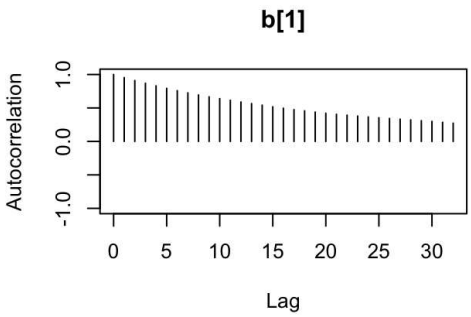

```
gelman.diag(mod1_sim)
```

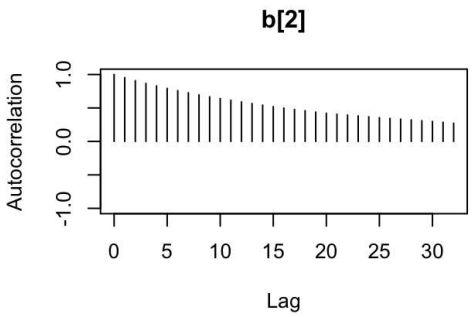
```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## b[1]          1          1
## b[2]          1          1
## sig           1          1
##
## Multivariate psrf
##
## 1
```

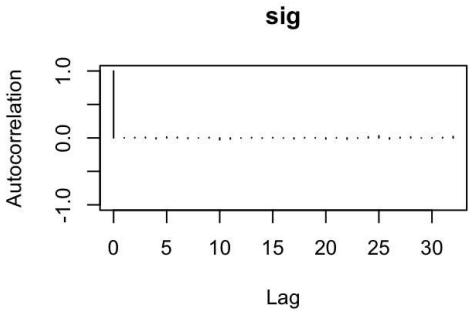
```
autocorr.diag(mod1_sim)
```

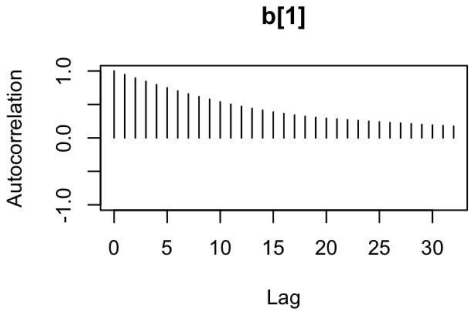
```
##              b[1]       b[2]          sig
## Lag 0   1.00000000 1.00000000  1.000000000
## Lag 1   0.95058437 0.95075735  0.013408884
## Lag 5   0.77384166 0.77462905  0.018885943
## Lag 10  0.59735920 0.59925640 -0.010108865
## Lag 50  0.08252587 0.08242905 -0.007541658
```

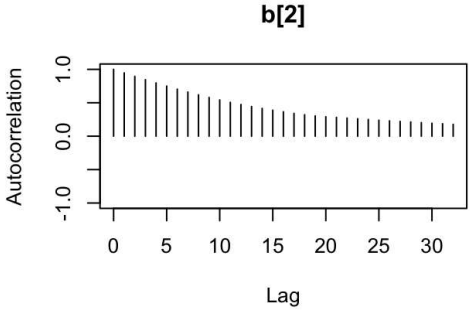
```
autocorr.plot(mod1_sim)
```

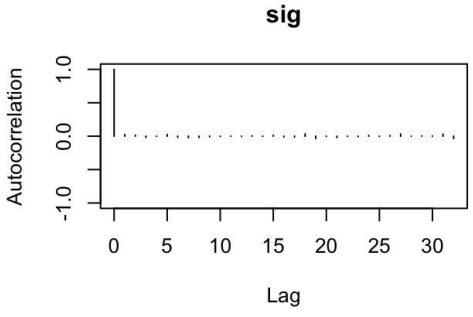

b[1]
Autocorrelation
1.0
0.0
-1.0
0
5
10
15
20
25
30
Lag

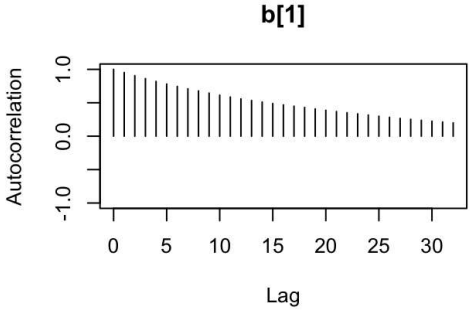

b[2]
Autocorrelation
1.0
0.0
-1.0
0
5
10
15
20
25
30
Lag

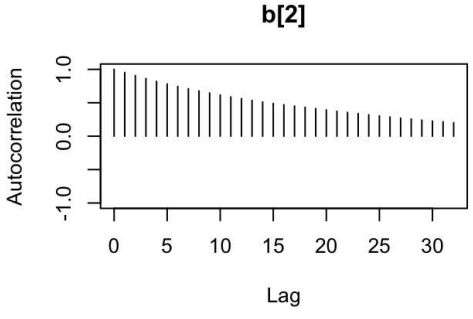

sig
Autocorrelation
1.0
0.0
-1.0
0
5
10
15
20
25
30
Lag

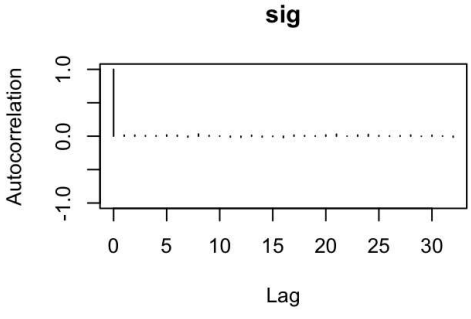

b[1]
Autocorrelation
1.0
0.0
-1.0
0
5
10
15
20
25
30
Lag


b[2]
Autocorrelation
1.0
0.0
-1.0
0
5
10
15
20
25
30
Lag


sig
Autocorrelation
1.0
0.0
-1.0
0
5
10
15
20
25
30
Lag


b[1]
Autocorrelation
1.0
0.0
-1.0
0
5
10
15
20
25
30
Lag


b[2]
Autocorrelation
1.0
0.0
-1.0
0
5
10
15
20
25
30
Lag


sig
Autocorrelation
1.0
0.0
-1.0
0
5
10
15
20
25
30
Lag

```
effectiveSize(mod1_sim)
```

```
##       b[1]       b[2]        sig
##   379.9761   373.4897 14768.4538
```

We can get a posterior summary of the parameters in our model.

```
summary(mod1_sim)
```

```
##
## Iterations = 1001:6000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##         Mean      SD  Naive SE Time-series SE
## b[1]  7.1305 0.43930 0.0035868      0.0226253
## b[2] -0.5092 0.07104 0.0005800      0.0036938
## sig   0.9712 0.06838 0.0005583      0.0005628
##
## 2. Quantiles for each variable:
##
##         2.5%     25%     50%     75%   97.5%
## b[1]  6.2820  6.8336  7.1273  7.4285  7.9984
## b[2] -0.6494 -0.5575 -0.5087 -0.4612 -0.3708
## sig   0.8480  0.9231  0.9672  1.0146  1.1155
```

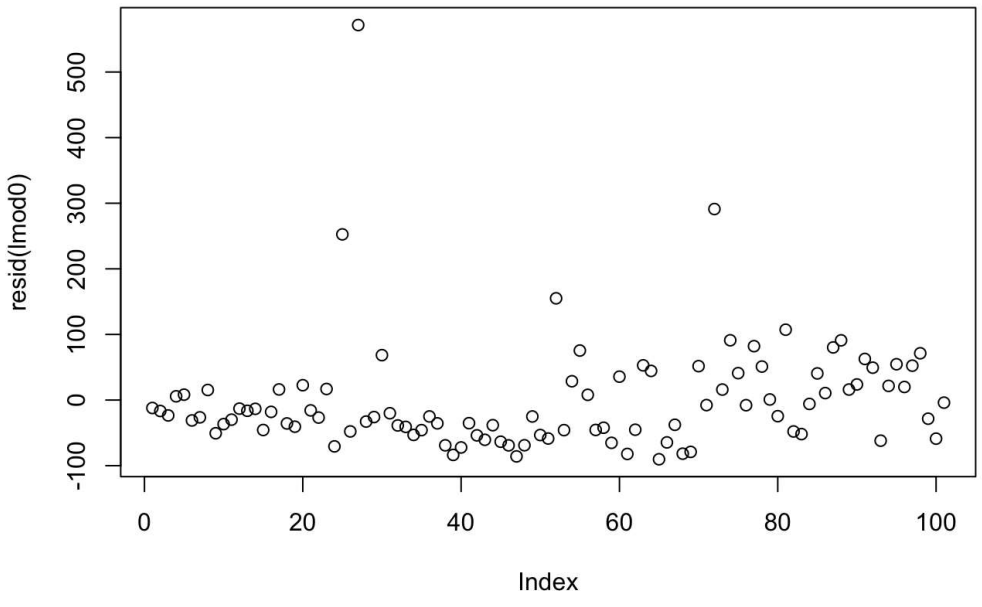
Don't forget that these results are for a regression model relating the logarithm of infant mortality to the logarithm of income.

# Residual checks

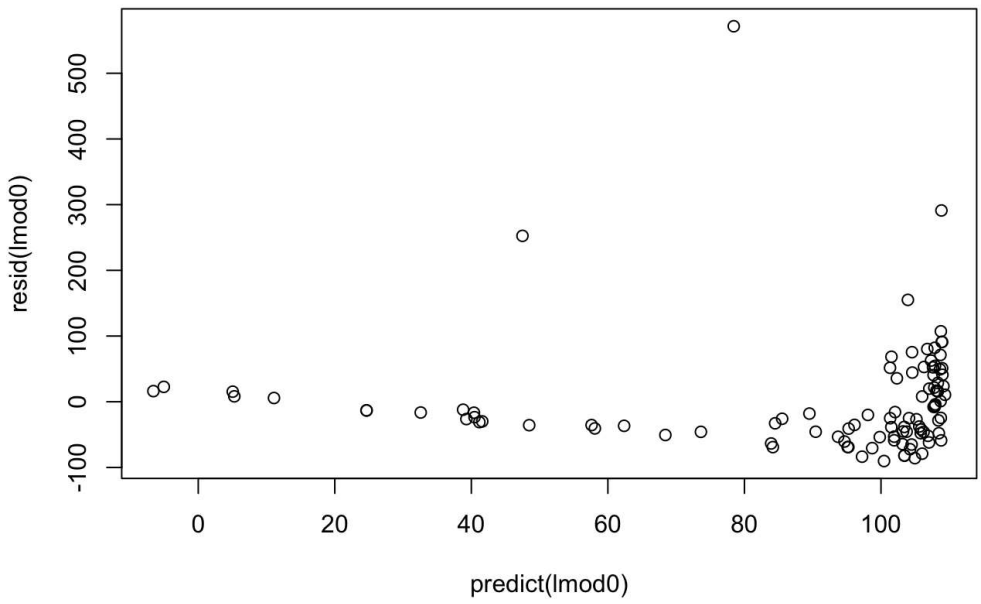
Checking residuals (the difference between the response and the model's prediction for that value) is important with linear models since residuals can reveal violations of the assumptions we made to specify the model. In particular, we are looking for any sign that the model is not linear, normally distributed, or that the observations are not independent (conditional on covariates).

First, let's look at what would have happened if we fit the reference linear model to the un-transformed variables.

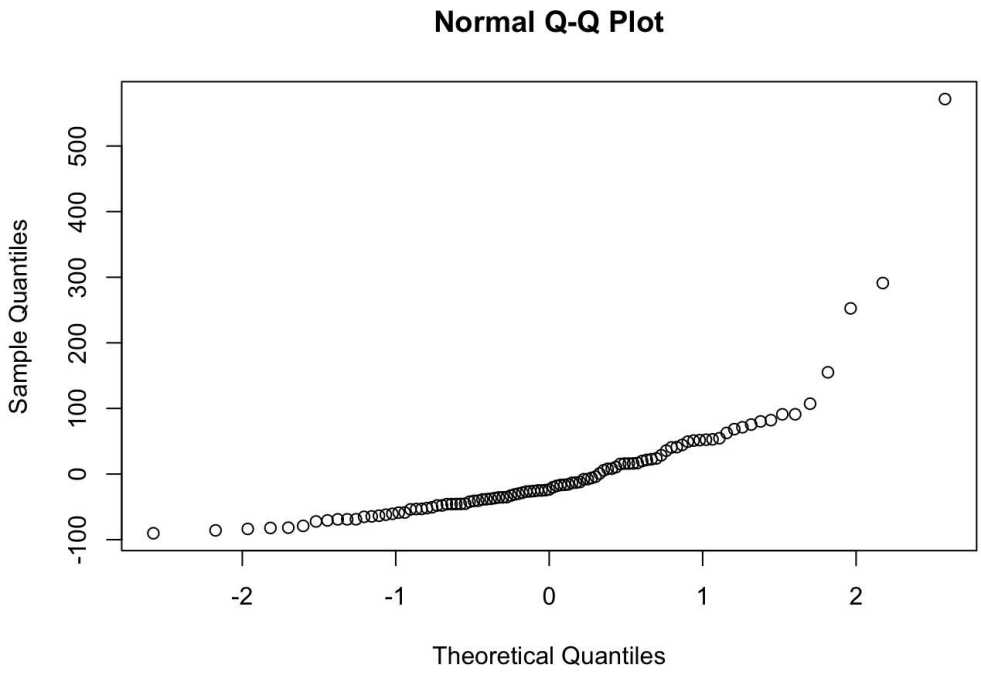
```
lmod0 = lm(infant ~ income, data=Leinhardt)
plot(resid(lmod0)) # to check independence (looks okay)
```




```
plot(predict(lmod0), resid(lmod0)) # to check for linearity, constant variance (looks bad)
```




```
qqnorm(resid(lmod0)) # to check Normality assumption (we want this to be a straight line)
```




Now let's return to our model fit to the log-transformed variables. In a Bayesian model, we have distributions for residuals, but we'll simplify and look only at the residuals evaluated at the posterior mean of the parameters.

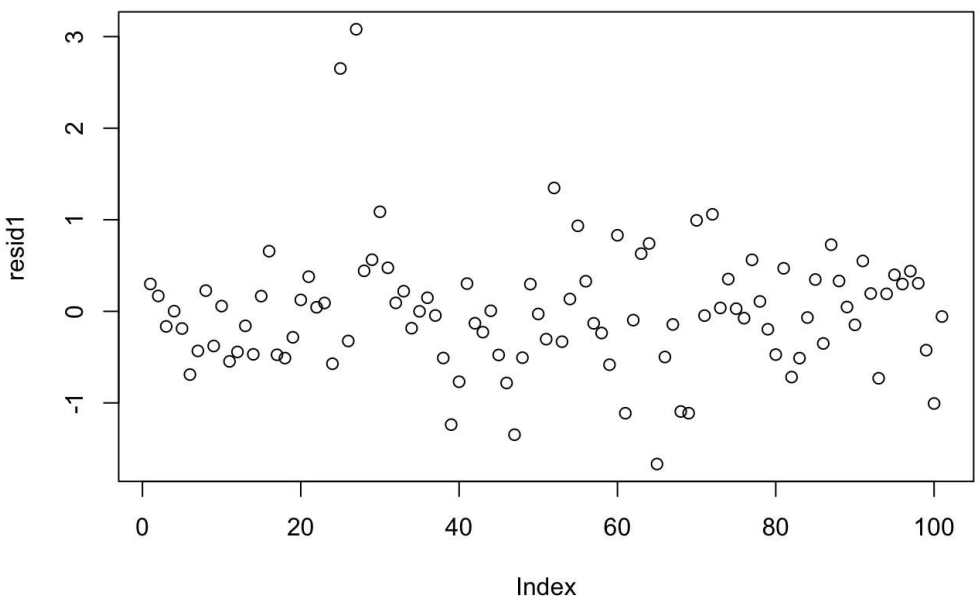
```
X = cbind(rep(1.0, data1_jags$n), data1_jags$log_income)
head(X)
```

```
##      [,1]     [,2]
## [1,]    1 8.139149
## [2,]    1 8.116716
## [3,]    1 8.115521
## [4,]    1 8.466110
## [5,]    1 8.522976
## [6,]    1 8.105308
```

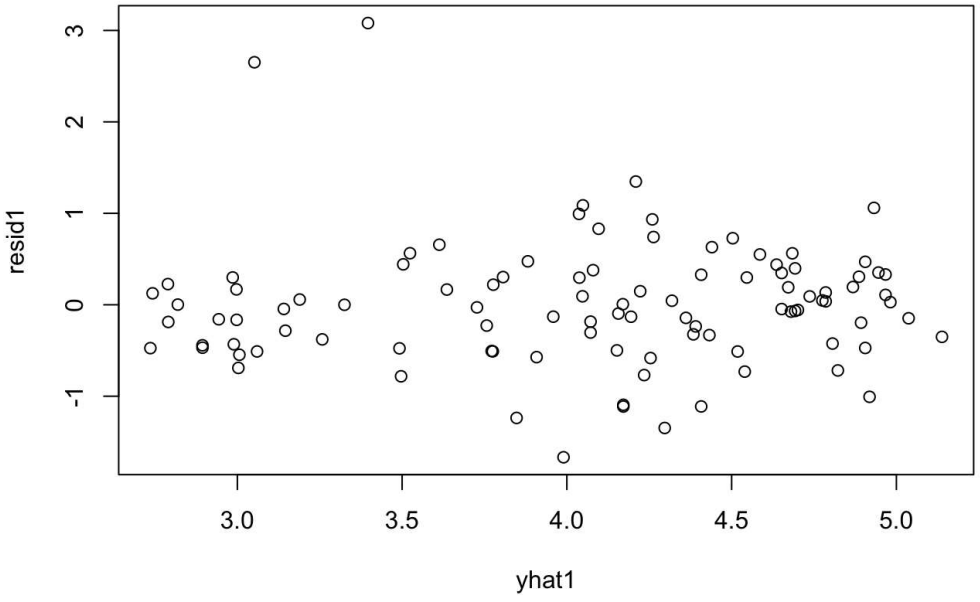
```
(pm_params1 = colMeans(mod1_csim)) # posterior mean
```

```
##       b[1]       b[2]        sig
##  7.1304729 -0.5092172  0.9711596
```

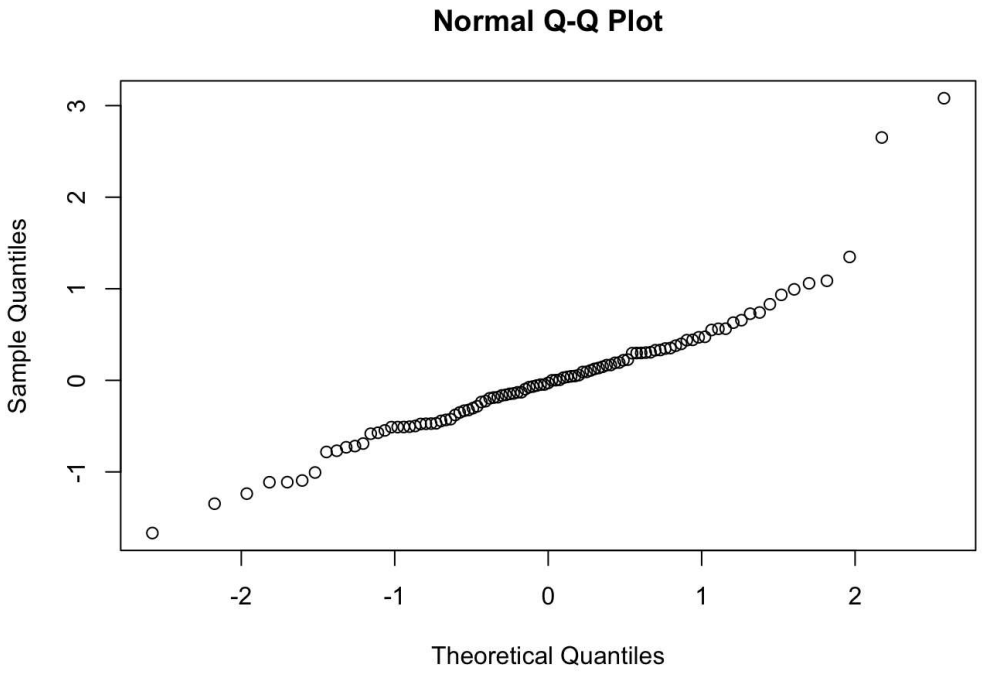
```
yhat1 = drop(X %*% pm_params1[1:2])
resid1 = data1_jags$y - yhat1
plot(resid1) # against data index
```



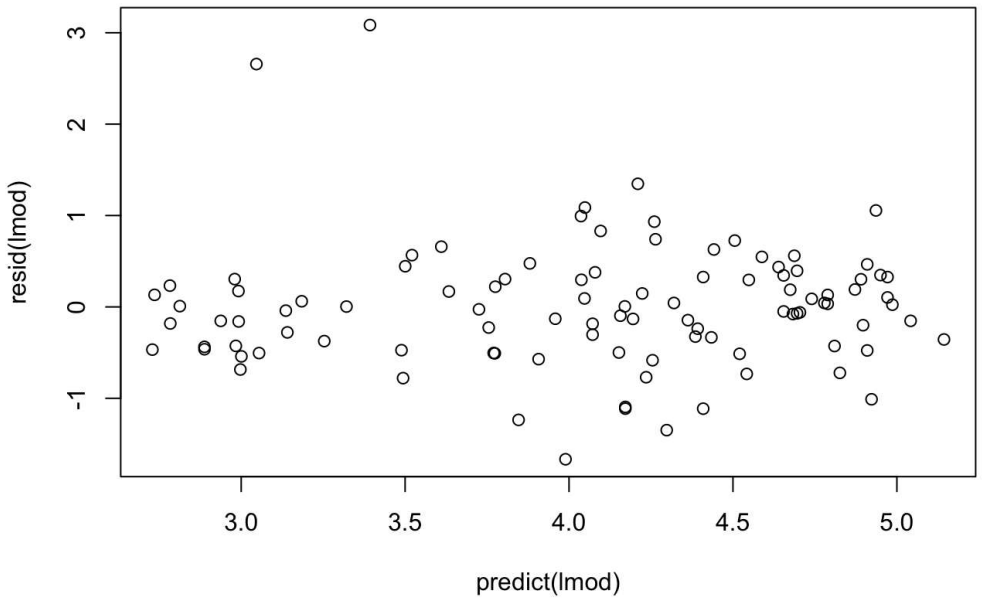

```
plot(yhat1, resid1) # against predicted values
```




```
qqnorm(resid1) # checking normality of residuals
```




```
plot(predict(lmod), resid(lmod)) # to compare with reference linear model
```




```
rownames(dat)[order(resid1, decreasing=TRUE)[1:5]] # which countries have the largest positive residuals?
```

```
## [1] "Saudi.Arabia" "Libya"        "Zambia"       "Brazil"
## [5] "Afganistan"
```

The residuals look pretty good here (no patterns, shapes) except for two strong outliers, Saudi Arabia and Libya. When outliers appear, it is a good idea to double check that they are not just errors in data entry. If the values are correct, you may reconsider whether these data points really are representative of the data you are trying to model. If you conclude that they are not (for example, they were recorded on different years), you may be able to justify dropping these data points from the data set.

If you conclude that the outliers are part of data and should not be removed, we have several modeling options to accommodate them. We will address these in the next segment.

# Lesson 7.5

In the previous segment, we saw two outliers in the model relating the logarithm of infant mortality to the logarithm of income. Here we will discuss options for when we conclude that these outliers belong in the data set.

## Additional covariates

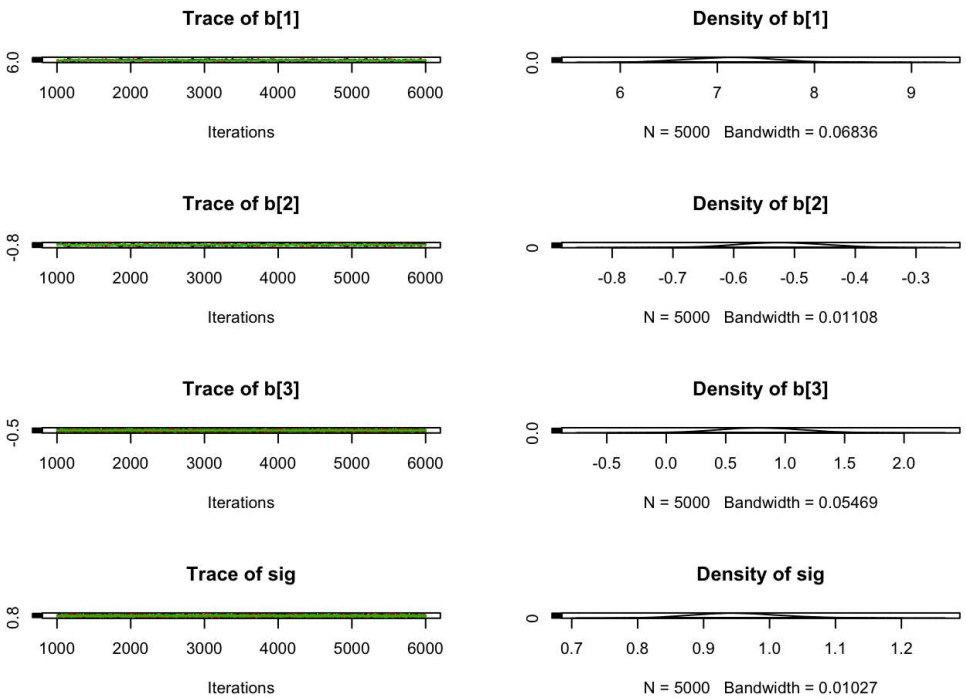
The first approach is to look for additional covariates that may be able to explain the outliers. For example, there could be a number of variables that provide information about infant mortality above and beyond what income provides.

Looking back at our data, there are two variables we haven't used yet: `region` and `oil`. The `oil` variable indicates oil-exporting countries. Both Saudi Arabia and Libya are oil-exporting countries, so perhaps this might explain part of the anomaly.

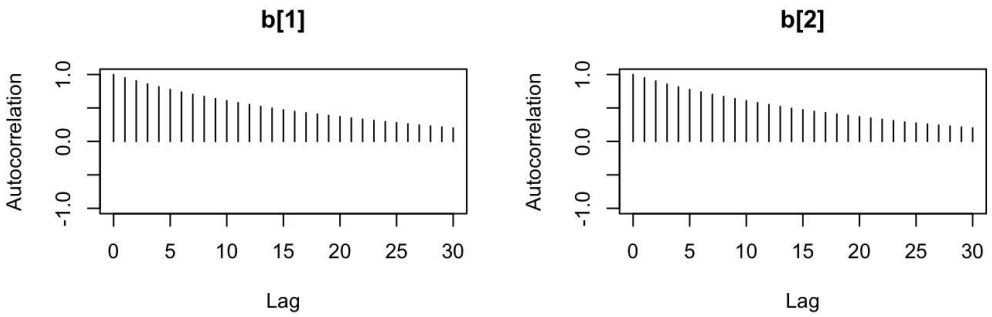
```
library("rjags")

mod2_string = " model {
    for (i in 1:length(y)) {
        y[i] ~ dnorm(mu[i], prec)
        mu[i] = b[1] + b[2]*log_income[i] + b[3]*is_oil[i]
    }

    for (i in 1:3) {
        b[i] ~ dnorm(0.0, 1.0/1.0e6)
    }

    prec ~ dgamma(5/2.0, 5*10.0/2.0)
    sig = sqrt( 1.0 / prec )
} "


set.seed(73)
data2_jags = list(y=dat$loginfant, log_income=dat$logincome,
                  is_oil=as.numeric(dat$oil=="yes"))
data2_jags$is_oil

params2 = c("b", "sig")

inits2 = function() {
    inits = list("b"=rnorm(3,0.0,100.0), "prec"=rgamma(1,1.0,1.0))
}

mod2 = jags.model(textConnection(mod2_string), data=data2_jags, inits=inits2, n.chains=3)
update(mod2, 1e3) # burn-in

mod2_sim = coda.samples(model=mod2,
                        variable.names=params2,
                        n.iter=5e3)

mod2_csim = as.mcmc(do.call(rbind, mod2_sim)) # combine multiple chains
```

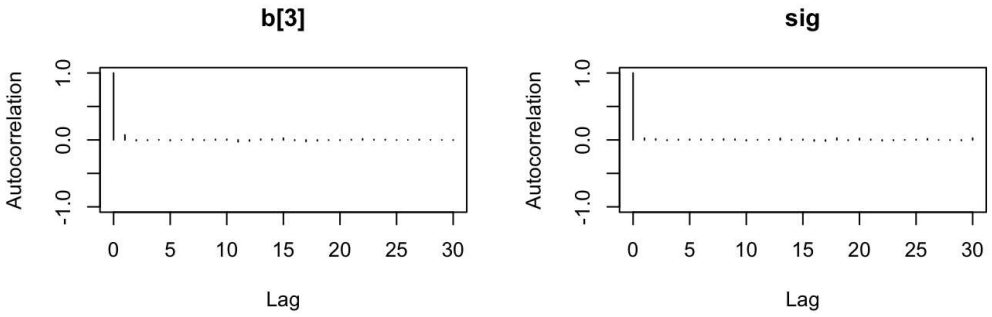
As usual, check the convergence diagnostics.

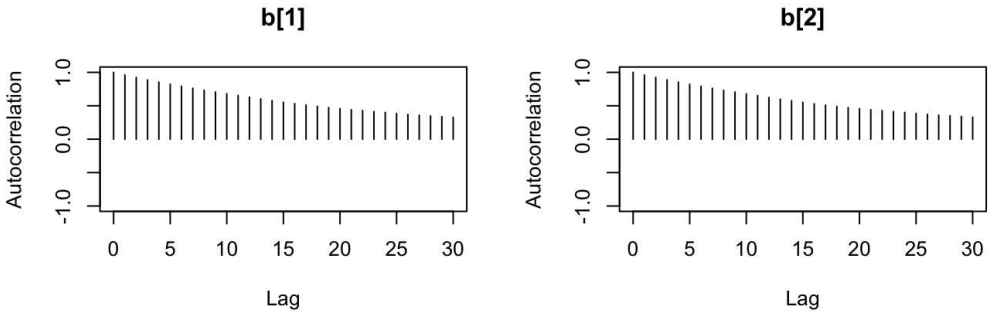
```
plot(mod2_sim)
```

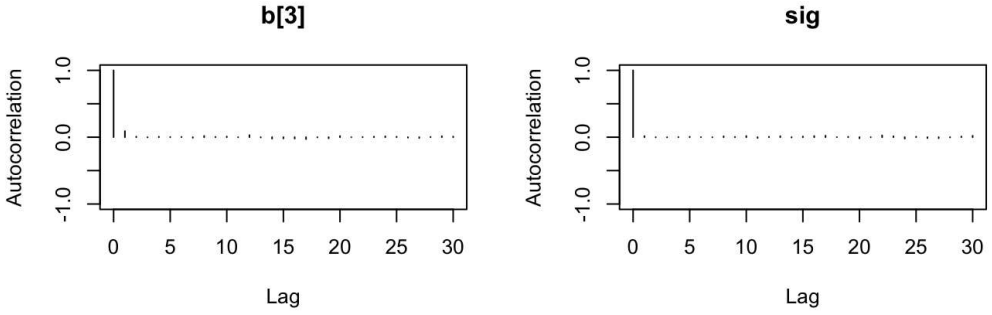


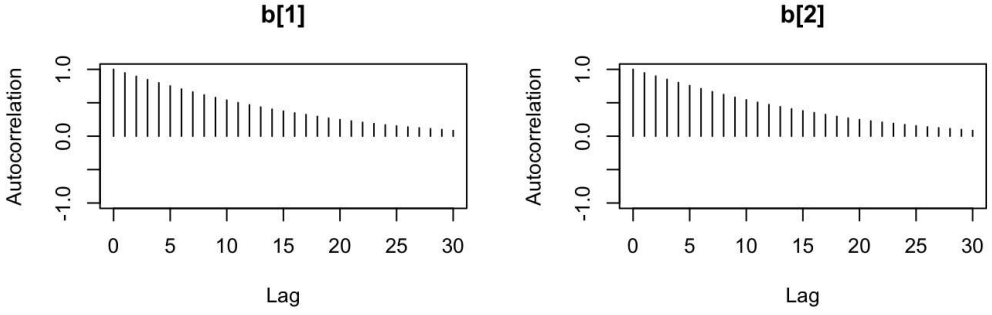

```
gelman.diag(mod2_sim)
autocorr.diag(mod2_sim)

autocorr.plot(mod2_sim)
```

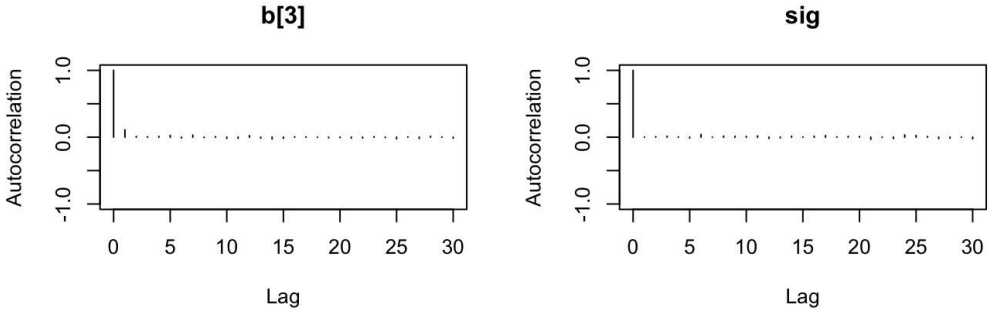

b[1]
b[2]
Autocorrelation
-1.0
0.0
1.0
0
5
10
15
20
25
30
Lag


b[3]
sig
Autocorrelation
-1.0
0.0
1.0
0
5
10
15
20
25
30
Lag


b[1]
b[2]
Autocorrelation
-1.0
0.0
1.0
0
5
10
15
20
25
30
Lag


b[3]
sig
Autocorrelation
-1.0
0.0
1.0
0
5
10
15
20
25
30
Lag


b[1]
b[2]
Autocorrelation
-1.0
0.0
1.0
0
5
10
15
20
25
30
Lag


b[3]
sig
Autocorrelation
-1.0
0.0
1.0
0
5
10
15
20
25
30
Lag

```
effectiveSize(mod2_sim)
```

We can get a posterior summary of the parameters in our model.

```
summary(mod2_sim)
```

```
##
## Iterations = 1001:6000
## Thinning interval = 1
## Number of chains = 3
## Sample size per chain = 5000
##
## 1. Empirical mean and standard deviation for each variable,
##    plus standard error of the mean:
##
##         Mean      SD  Naive SE Time-series SE
## b[1]  7.1443 0.44131 0.0036033      0.0237479
## b[2] -0.5219 0.07152 0.0005840      0.0038277
## b[3]  0.7902 0.35301 0.0028823      0.0031607
## sig   0.9527 0.06744 0.0005506      0.0005557
##
## 2. Quantiles for each variable:
##
##          2.5%     25%     50%     75%   97.5%
## b[1]  6.29260  6.8406  7.1452  7.4418  8.0050
## b[2] -0.66269 -0.5701 -0.5223 -0.4732 -0.3832
## b[3]  0.09124  0.5552  0.7887  1.0291  1.4800
## sig   0.82979  0.9066  0.9490  0.9954  1.0947
```

It looks like there is a positive relationship between oil-production and log-infant mortality. Because these data are merely observational, we cannot say that oil-production causes an increase in infant mortality (indeed that most certainly isn't the case), but we can say that they are positively correlated.

Now let's check the residuals.

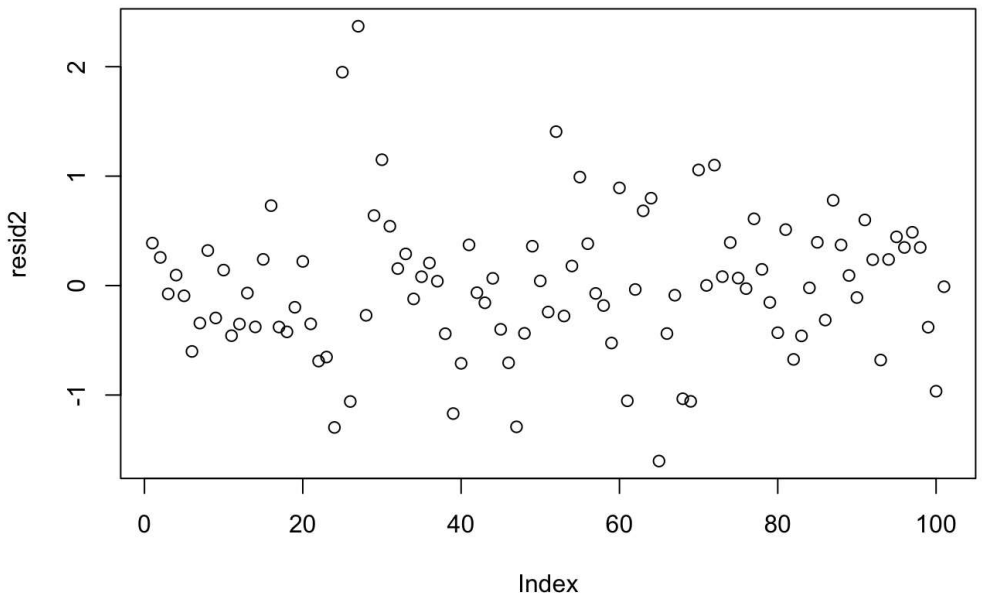
```
X2 = cbind(rep(1.0, data1_jags$n), data2_jags$log_income, data2_jags$is_oil)
head(X2)
```

```
##      [,1]     [,2] [,3]
## [1,]    1 8.139149    0
## [2,]    1 8.116716    0
## [3,]    1 8.115521    0
## [4,]    1 8.466110    0
## [5,]    1 8.522976    0
## [6,]    1 8.105308    0
```

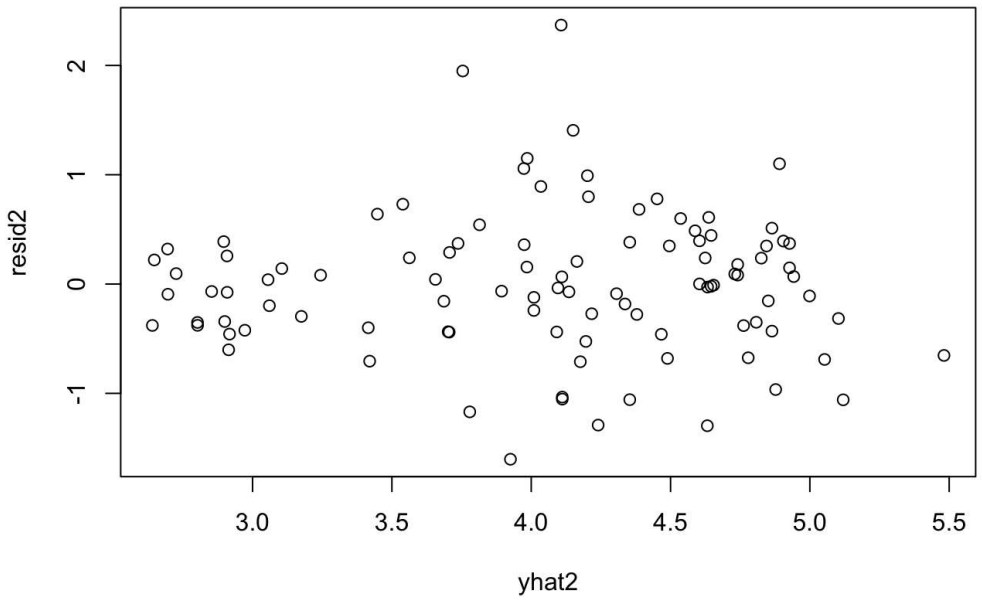
```
(pm_params2 = colMeans(mod2_csim)) # posterior mean
```

```
##       b[1]       b[2]       b[3]        sig
##  7.1442904 -0.5219009  0.7902475  0.9526686
```

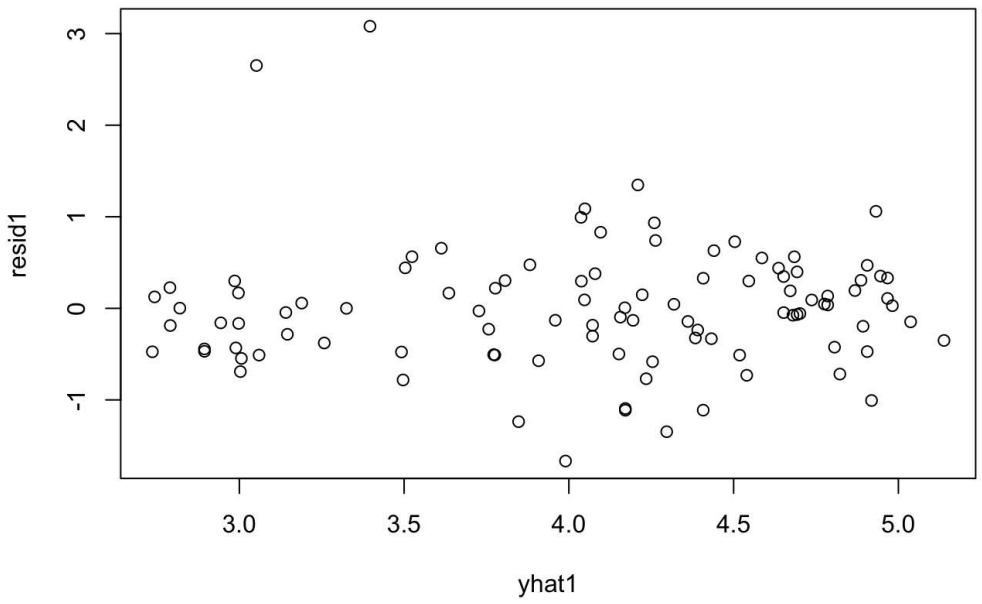
```
yhat2 = drop(X2 %*% pm_params2[1:3])
resid2 = data2_jags$y - yhat2
plot(resid2) # against data index
```




```
plot(yhat2, resid2) # against predicted values
```




```
plot(yhat1, resid1) # residuals from the first model
```




```
sd(resid2) # standard deviation of residuals
```

```
## [1] 0.6488704
```

These look much better, although the residuals for Saudi Arabia and Libya are still more than three standard deviations away from the mean of the residuals. We might consider adding the other covariate `region`, but instead let's look at another option when we are faced with strong outliers.

## $t$ likelihood

Let's consider changing the likelihood. The normal likelihood has thin tails (almost all of the probability is concentrated within the first few standard deviations from the mean). This does not accommodate outliers well. Consequently, models with the normal likelihood might be overly-influenced by outliers. Recall that the $t$ distribution is similar to the normal distribution, but it has thicker tails which can accommodate outliers.

The $t$ linear model might look something like this. Notice that the $t$ distribution has three parameters, including a positive "degrees of freedom" parameter. The smaller the degrees of freedom, the heavier the tails of the distribution. We might fix the degrees of freedom to some number, or we can assign it a prior distribution.

```
mod3_string = " model {
    for (i in 1:length(y)) {
        y[i] ~ dt( mu[i], tau, df )
        mu[i] = b[1] + b[2]*log_income[i] + b[3]*is_oil[i]
    }

    for (i in 1:3) {
        b[i] ~ dnorm(0.0, 1.0/1.0e6)
    }

    df = nu + 2.0 # we want degrees of freedom > 2 to guarantee existence of mean and variance
    nu ~ dexp(1.0)

    tau ~ dgamma(5/2.0, 5*10.0/2.0) # tau is close to, but not equal to the precision
    sig = sqrt( 1.0 / tau * df / (df - 2.0) ) # standard deviation of errors
} "
```

We will leave it up to you to fit this model.

# Lesson 7.6

## Compare models using Deviance Information Criterion

We have now proposed three different models. How do we compare their performance on our data? In the previous course, we discussed estimating parameters in models using the maximum likelihood method. Similarly, we can choose between competing models using the same idea.

We will use a quantity known as the deviance information criterion (DIC). It essentially calculates the posterior mean of the log-likelihood and adds a penalty for model complexity.

Let's calculate the DIC for our first two models:

the simple linear regression on log-income,

```
dic.samples(mod1, n.iter=1e3)
```

```
## Mean deviance:  231.2
## penalty 2.911
## Penalized deviance: 234.1
```

and the second model where we add oil production.

```
dic.samples(mod2, n.iter=1e3)
```

```
## Mean deviance:  225.4
## penalty 4.006
## Penalized deviance: 229.4
```

The first number is the Monte Carlo estimated posterior mean deviance, which equals $-2$ times the log-likelihood (plus a constant that will be irrelevant for comparing models). Because of that $-2$ factor, a smaller deviance means a higher likelihood.

Next, we are given a penalty for the complexity of our model. This penalty is necessary because we can always increase the likelihood of the model by making it more complex to fit the data exactly. We don't want to do this because over-fit models generalize poorly. This penalty is roughly equal to the effective number of parameters in your model. You can see this here. With the first model, we had a variance parameter and two betas, for a total of three parameters. In the second model, we added one more beta for the oil effect.

We add these two quantities to get the DIC (the last number). The better-fitting model has a lower DIC value. In this case, the gains we receive in deviance by adding the `is_oil` covariate outweigh the penalty for adding an extra parameter. The final DIC for the second model is lower than for the first, so we would prefer using the second model.

We encourage you to explore different model specifications and compare their fit to the data using DIC. Wikipedia (https://en.wikipedia.org/wiki/Deviance_information_criterion) provides a good introduction to DIC and we can find more details about the `JAGS` implementation through the `rjags` package documentation by entering `?dic.samples` in the `R` console.