Math 463 HW5

Ian McConachie

Teen Gambling Revisited

To start out with, we can use the R below to retrieve the teen gambling data that we are going to be fitting models to.

```r
myfile <- "https://pages.uoregon.edu/dlevin/DATA/teengamb.txt"
teengamb = read.table(myfile, header = T)
teengamb$sex <- as.factor(teengamb$sex) # changing sex into a categorical variable

# This for loop makes a new variable in the data frame for the log of amount gambled
# I had to put it in a for loop because of the errors with log of 0
log_gamble = 1:length(teengamb$gamble)
for (i in 1:length(teengamb$gamble)) {
  if ((teengamb$gamble[i]) == 0) {
    log_gamble[i] = 0
  } else {
    log_gamble[i] = log(teengamb$gamble[i])
  }
}
teengamb$log_G <- log_gamble
```

We can then generate the 5 different Bayesian regression models we fit in homework 4 using stan_glm.

```r
# This model has both sex and income as predictors with an interaction variable
g_fit1 = stan_glm(log_G~income*sex, data=teengamb, refresh=0)
# This model has income as a main effect with sex included in an interaction variable
g_fit2 = stan_glm(log_G~income + income:sex, data=teengamb, refresh=0)
# This model has sex as a main effect with income included only in an interaction variable
g_fit3 = stan_glm(log_G~sex + sex:income, data=teengamb, refresh=0)
# This model only has income as a predictor
g_fit4 = stan_glm(log_G~income, data=teengamb, refresh=0)
# This model only has sex as a predictor
g_fit5 = stan_glm(log_G~sex, data=teengamb, refresh=0)
```

Now we will use Leave-One-Out cross validation (LOO) to evaluate the predictive capability of all 5 of these models—this kind of cross validation can be done automatically with the loo() function in R. When we generate all the LOO summaries with the below code, we want to look for the fit that has the largest expected log probability density (ELPD) because ELPD = -1/2(sum(y_i - phi_i)^2). So, the larger the EPLD the smaller the sum of the predictive errors (sum(y_i - phi_i)^2)) is.

```r
loo(g_fit1)
```

```
##
## Computed from 4000 by 47 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo    -93.1 4.6
## p_loo         4.6 1.1
## looic       186.2 9.1
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     46    97.9%   1079
##  (0.5, 0.7]   (ok)        1     2.1%   377
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
loo(g_fit2)
```

```
##
## Computed from 4000 by 47 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo    -92.5 4.5
## p_loo         4.0 1.0
## looic       184.9 9.0
## ------
## Monte Carlo SE of elpd_loo is NA.
##
## Pareto k diagnostic values:
##                          Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     45    95.7%   1326
##  (0.5, 0.7]   (ok)        1     2.1%   1111
##    (0.7, 1]   (bad)       1     2.1%   814
##    (1, Inf)   (very bad)  0     0.0%   <NA>
## See help('pareto-k-diagnostic') for details.
```

```
loo(g_fit3)
```

```
##
## Computed from 4000 by 47 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo    -93.3 4.5
## p_loo         4.8 1.2
## looic       186.7 9.1
## ------
## Monte Carlo SE of elpd_loo is 0.1.
##
## Pareto k diagnostic values:
```

```
##                         Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     45    95.7%   1397
##  (0.5, 0.7]   (ok)        2     4.3%   269
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
loo(g_fit4)
```

```
##
## Computed from 4000 by 47 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo    -98.0  5.4
## p_loo         2.8  0.9
## looic       196.0 10.7
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## Pareto k diagnostic values:
##                         Count Pct.    Min. n_eff
## (-Inf, 0.5]   (good)     46    97.9%   1946
##  (0.5, 0.7]   (ok)        1     2.1%   644
##    (0.7, 1]   (bad)       0     0.0%   <NA>
##    (1, Inf)   (very bad)  0     0.0%   <NA>
##
## All Pareto k estimates are ok (k < 0.7).
## See help('pareto-k-diagnostic') for details.
```

```
loo(g_fit5)
```

```
##
## Computed from 4000 by 47 log-likelihood matrix
##
##          Estimate  SE
## elpd_loo    -96.4 4.3
## p_loo         2.6 0.6
## looic       192.8 8.6
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

The elpd_loo values above will vary each time this code is run because of the pseduo-random processes involved in the Bayesian model generator stan_glm. However, every time I have run this code, I have ended up with the second model (g_fit2) having the lowest elpd_loo value. This suggests that the second model has the greatest predictive capability and therefore is the model we should favor when predicting values in this data.

However, it is important to note that the standard errors of the elpd_loo values for every one of these models is large enough so that the ELPD estimate for each of the other models is contained within one standard error of it—perhaps with the exception of g_fit4 which is beyond 1 standard error from the second model's estimate. This suggests that although thess models do have varying predictive capability, the variation is not so much to rule out the possibility that the difference was caused by random error.

Now that we have done our LOO analysis, we can do another round of cross validation in the form of K-fold cross-validation.

```
# Do some K-fold analysis on each of our models with the K value of 5
k_value = 5
kf1 <- kfold(g_fit1, K=k_value); kf2 <- kfold(g_fit2,K=k_value)
kf3 <- kfold(g_fit3, K=k_value); kf4 <- kfold(g_fit4, K=k_value)
kf5 <- kfold(g_fit5, K=k_value)
# Print out the results of our cross validation
print(kf1); print(kf2); print(kf3); print(kf4); print(kf5)
```

```
##
## Based on 5-fold cross-validation
##
##            Estimate  SE
## elpd_kfold    -92.7 4.4
## p_kfold          NA  NA
## kfoldic       185.4 8.7

##
## Based on 5-fold cross-validation
##
##            Estimate  SE
## elpd_kfold    -92.9 4.4
## p_kfold          NA  NA
## kfoldic       185.9 8.8

##
## Based on 5-fold cross-validation
##
##            Estimate  SE
## elpd_kfold    -93.6 4.7
## p_kfold          NA  NA
## kfoldic       187.2 9.4

##
## Based on 5-fold cross-validation
##
##            Estimate   SE
## elpd_kfold    -99.0  5.3
## p_kfold          NA   NA
## kfoldic       197.9 10.6

##
## Based on 5-fold cross-validation
##
##            Estimate  SE
## elpd_kfold    -95.1 4.0
## p_kfold          NA  NA
## kfoldic       190.3 8.1
```

Each time we run the above code, the k-fold function uses pseudo-random numbers to select the K groups, so each time this code is run there will be different results, just like above. However, after running it several times, I have noticed that the results of this analysis are essentially the same as the LOO analysis we did above. The second model generally has the highest ELPD value suggesting it is the model with the best predictive ability— but the standard errors do not necessarily ensure that this difference in ELPD is not caused by random error.

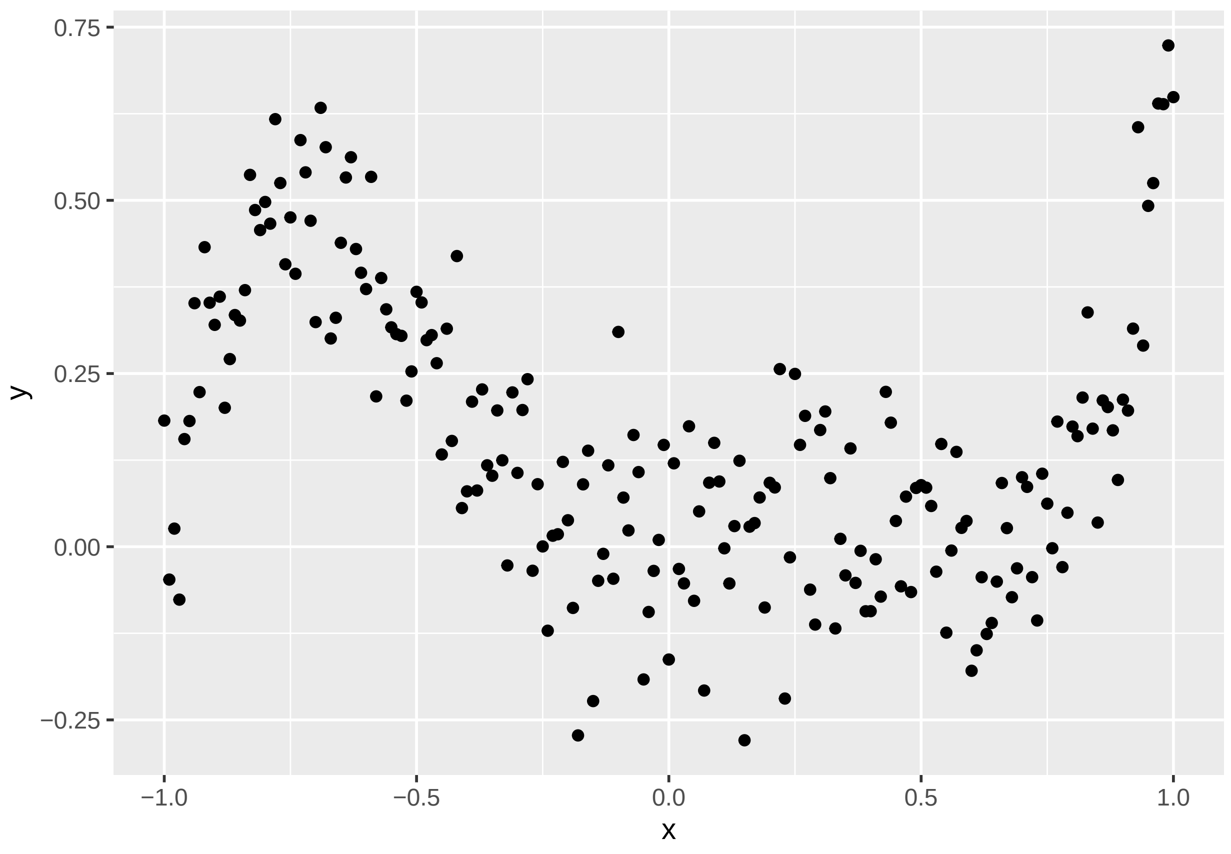
Polynomial Fitting

First, we can use the below code to import the data from the uoregon server where it is hosted.

```r
myfile <- "https://pages.uoregon.edu/dlevin/DATA/xyfinal19.txt"
xy = read.table(myfile, header = T)
```

Now, before we do any of our regression analysis, we can plot the points and see the shape of the data we are working with.

```r
ggplot(xy, aes(x=x,y=y)) + geom_point()
```



In order to do a simple cross validation, we want to split our data into a training set and a validation set. The training set will be used to build our model and then the validation set will be used to test the predictive ability of the model.

```
n = dim(xy)[1]
# This line will generate a random sample of datapoints for our test data set
itest <- sample(1:n, 50, replace=FALSE)
xytest <- xy[itest,]
xytrain <- xy[-itest,]
```

Now that we have split our data, we can build Bayesian linear regression models with each predictor being a different power of x to simulate a polynomial function using our linear combination. We can test different degrees of polynomials from 1 to 8, generating a model with the training data for each and then testing it against the validation data. While we are doing the standard split cross validation, we can generate a cross validation analysis using leave-one-out (LOO) in the same for loop to knock two birds out with one stone.

```
# Set up the vectors we will input data into
pes <- 1:8
loo_epld <- 1:8
k_epld <- 1:8
# Do cross validation on each of our 8 models
for (i in 1:8) {
  f <- stan_glm(y~poly(x,i), data=xytrain, refresh=0)
  f_full <- stan_glm(y~poly(x,i), data=xy, refresh=0)
  # Standard cross validation
  pf <- predict(f, newdata=xytest)
  pes[i] = sum((xytest$y - pf)^2)
  # LOO cross validation
  loo_epld[i] <- loo(f_full)$estimate[1,1]
}
print("Here are our predictive error estimates based on standard cross validation")
```

```
## [1] "Here are our predictive error estimates based on standard cross validation"
```

```
print(pes)
```

```
## [1] 1.2806962 1.3621671 0.8855643 0.8253878 0.6244952 0.6152213 0.6103508
## [8] 0.6273983
```

```
print("Here are our ELPD values calculated using LOO cross validation")
```

```
## [1] "Here are our ELPD values calculated using LOO cross validation"
```

```
print(loo_epld)
```

```
## [1]  36.83035  71.05775 105.02277 106.30457 155.69906 155.14924 154.55956
## [8] 153.58131
```

For the predictive error estimates, we want to choose the model with the lowest value because this value represents the amount of error the model exhibits when testing it against the validation set. For the ELPD values, we want the model with the highest value because the EPLD involves multiplying the error term by a negative number.

The results of the above cross validation analysis of these 8 models will vary each time because of the stochastic processes involved in stan_glm. However, after running this code several times, I have consistently seen that the 5th model (corresponding to a 5th degree polynomial) gives us the largest ELPD estimates and the smallest predictive error estimates.

Often, among the predictive error estimates, there are higher degree polynomials that have similar estimate values [this also sometimes happens with the EPLD estimates]. In these situations, we can note that for the smaller degree polynomials the predictive capability is generally substantially less than the 5th degree model which suggests that we need at least a 5th degree model. When there are larger models with similar or the same predictive capability, we always want to pick the smallest model (one which the other models contain). This choice can be intuitively justified with Occam's Razor: the idea that the simpler answer to a question is generally the correct one. From a statistical perspective this can be justified by the tendency of larger models to overfit to noise rather than actually increase the ability of the model to make predictions about future, unseen data.

Modeling Ozone

First we can write some R code to import the Ozone data that we will be fitting data to.

```
myfile <- "https://pages.uoregon.edu/dlevin/DATA/ozone.txt"
ozone = read.table(myfile, header = T)
```

Then we can generate 4 models to fit to this data with Ozone levels (given in the O3 variable) as our response and some combination of wind, temperature, and humidity as predictors. I chose to narrow down possible predictors to these three variables because they logically have an impact on the ozone levels and also they were the only variables in the data that I am certain of their meanings.

```
# Temp and humidity as predictors
ofit1 = stan_glm(O3~temp + humidity, data=ozone, refresh=0)
# Temp and humidity as predictors with an interaction variable
ofit2 = stan_glm(O3~temp*humidity, data=ozone, refresh=0)
# Temp, humidity, and wind as predictors
ofit3 = stan_glm(O3~temp + humidity + wind, data=ozone, refresh=0)
# Temp, humidity, and wind as predictors
ofit4 = stan_glm(O3~temp*humidity + wind, data=ozone, refresh=0)
```

Now, let's do some cross validation analysis on these 4 models to see if there exists a clear favorite in terms of predictive ability. For our cross validation, we'll use the loo function which automatically does leave-one-out cross validation.

```
loo(ofit1)
```

```
##
## Computed from 4000 by 330 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -986.3 13.2
## p_loo         3.9  0.4
## looic      1972.6 26.4
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```r
loo(ofit2)
```

```
##
## Computed from 4000 by 330 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -973.0 14.3
## p_loo         4.5  0.5
## looic      1946.0 28.6
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```r
loo(ofit3)
```

```
##
## Computed from 4000 by 330 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -986.3 13.0
## p_loo         4.5  0.4
## looic      1972.6 26.1
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

```r
loo(ofit4)
```

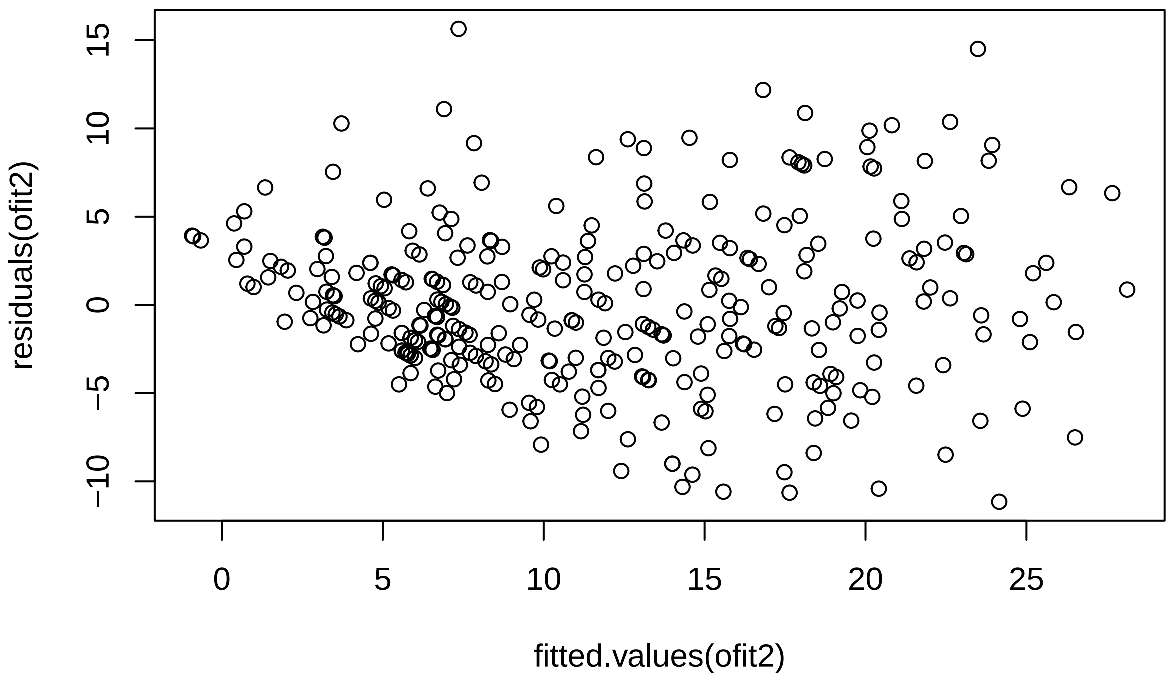
```
##
## Computed from 4000 by 330 log-likelihood matrix
##
##          Estimate   SE
## elpd_loo   -973.1 14.1
## p_loo         5.3  0.6
## looic      1946.1 28.1
## ------
## Monte Carlo SE of elpd_loo is 0.0.
##
## All Pareto k estimates are good (k < 0.5).
## See help('pareto-k-diagnostic') for details.
```

From the above analyses produced by the loo function for each model, we can compare the elpd_loo values between the model fits and select the one with the best ability to make predictions: the higher the elpd, the higher the predictive ability. Here, this would be the model that includes temperature, humidity as main effects with a variable for their interaction as well.

However, even though ofit2 has the best ELPD value of the group, it is important to note that the other ELPD values are all within one standard error's margin of its ELPD value implying that it is possible that it is not better at predicting than the other models, rather the values we observed came about because of random variation. This is important to consider, but ultimately this is the information we have to work with, so we can go ahead and use ofit2.

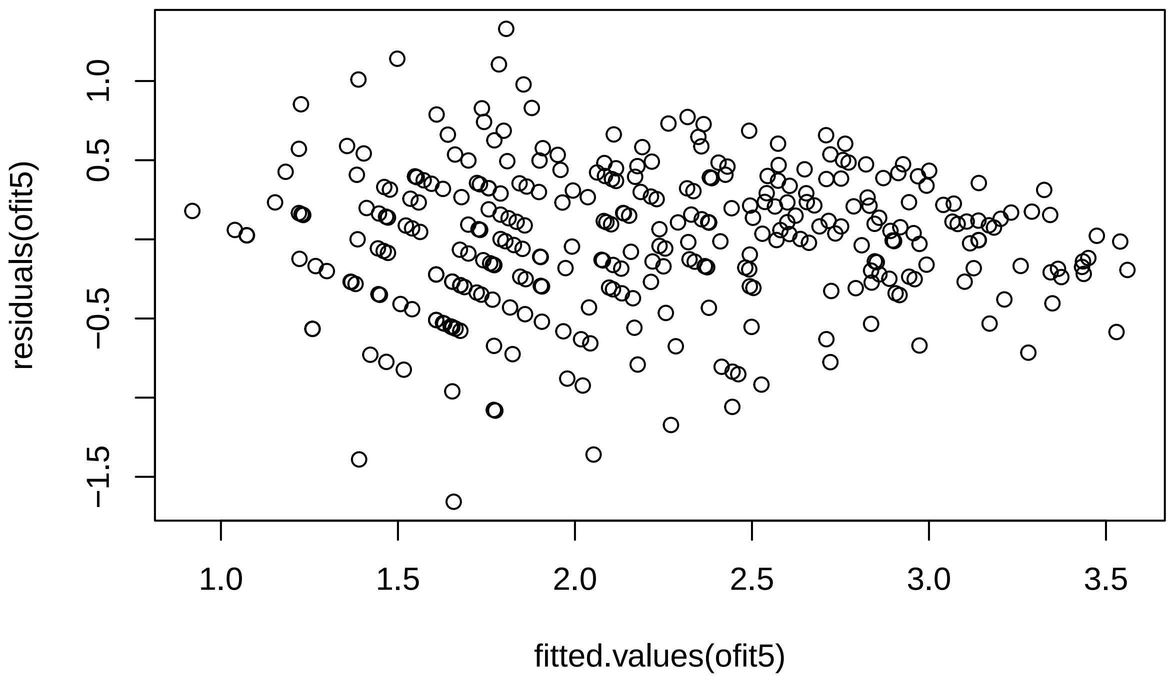
To visually examine the fit of this model, we can generate a residual plot to see if the residuals plotted against the fitted values results in a roughly balanced distribution in the x and y axises, as well as a roughly random distribution of residuals centered around the horizontal line 0.

```
plot(residuals(ofit2)~fitted.values(ofit2), data=ozone)
```



In the above residual plot, we can notice a pattern—specifically, the larger the fitted value, the more likely there will be a large residual. To correct this pattern and get a residual plot that is more in line with our assumptions about regression analysis, we can transform the response variable (ozone levels) by taking its log. The below R code generates such a model and then prints out its residual plot for visual inspection.

```
# Temp and humidity as predictors with an interaction variable
ofit5 = stan_glm(log(O3)~temp*humidity, data=ozone, refresh=0)
plot(residuals(ofit5)~fitted.values(ofit5), data=ozone)
```



The above plot does a good job satisfying everything we are looking for in a well-fitted model's residual plot: the x and y values are evenly spread and the residuals have no strong pattern. This tells us that taking the log of ozone levels likely resulted in a model more properly fit to the data and therefore is this the model we would favor.

11.9: Leave-one-out cross validation

Like the other problems, we can start off by importing the data with the following code.

```
myfile <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/Beauty/data/beauty.csv"
beauty <- read.csv(myfile, header = T)
beauty$female = as.factor(beauty$female)
beauty$nonenglish = as.factor(beauty$nonenglish)
sorted_beauty <- beauty[order(beauty$beauty),]
```

We can then fit a few models to this data with teacher evaluation as our response variable.

```
# Model 0: just age as a predictor
bfit0 = stan_glm(eval~age, data=sorted_beauty, refresh=0)
# Model 1: just beauty as a predictor
bfit1 = stan_glm(eval~beauty, data=sorted_beauty, refresh=0)
# Model 2: beauty and age as predictors
bfit2 = stan_glm(eval~beauty + age, data=sorted_beauty, refresh=0)
# Model 3: beauty and age as predictors with an interaction variable included
bfit3 = stan_glm(eval~beauty*age, data=sorted_beauty, refresh=0)
```

Now let's analyse the models above using leave-one-out (LOO) cross validation with the following code.

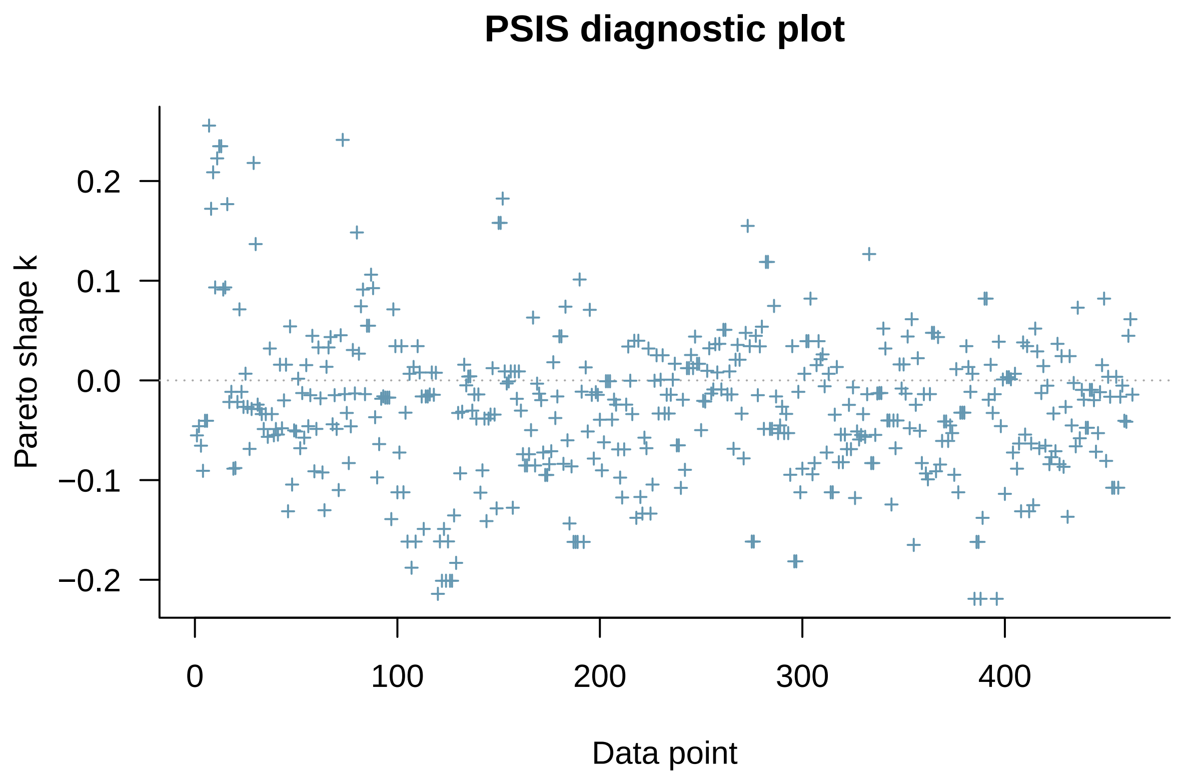
```
lbf0 = loo(bfit0, save_psis = TRUE)
lbf1 = loo(bfit1, save_psis = TRUE)
lbf2 = loo(bfit2, save_psis = TRUE)
lbf3 = loo(bfit3, save_psis = TRUE)
```

The results of the above cross validation will be different every time, but after running the code a few times, I have seen that the 4th model generally has the highest ELPD estimate. Higher ELPD estimates correspond to the models with more accurate predictions, so from this we would conclude that the 4th model (bfit3)—with beauty and age as predictors with both main effects and an interaction variable between the two—has the most accurate predictions and is therefore the model we should favor.

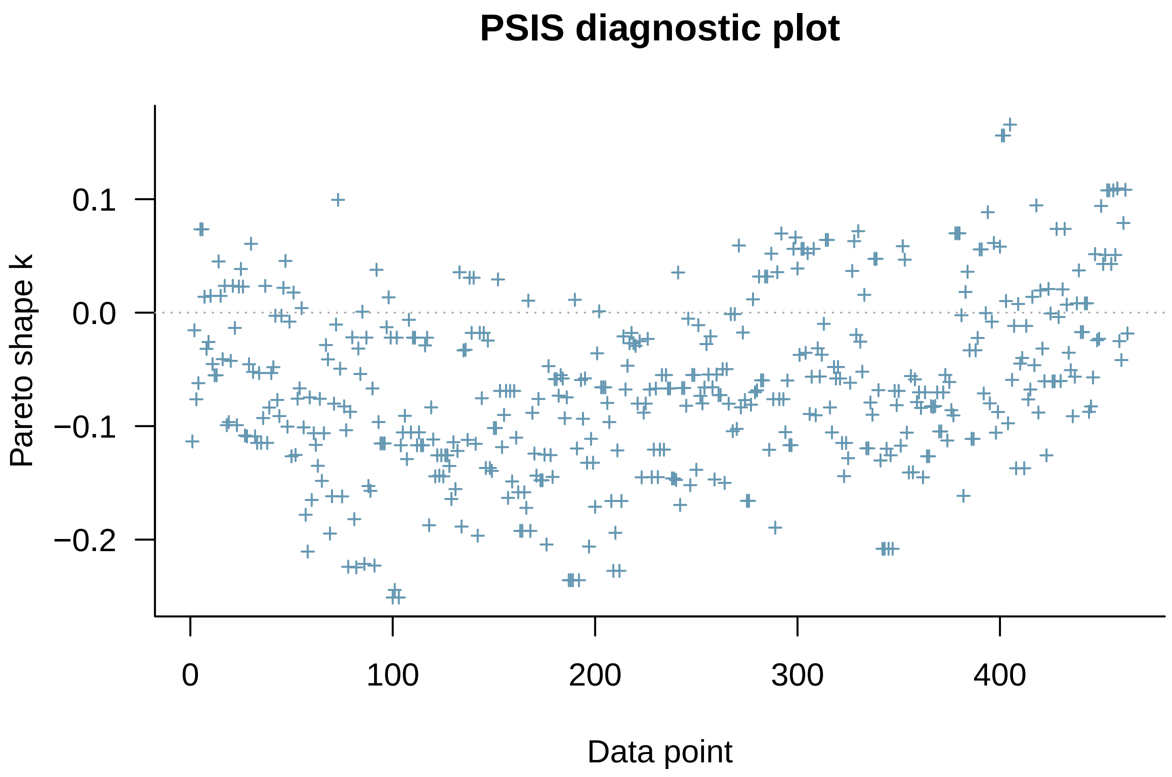
One important consideration for the analysis that we did above is that the standard errors of all the ELPD estimates are such that every other estimate is within one standard error from a given model's estimate. This tells us that other models' estimates are within a 68% confidence interval of any particular model's estimate and suggests that the difference in ELPD could be due to random variation rather than true difference in predictive accuracy. Despite this asterisk, bfit3 is still the model we should favor given this analysis.

We can plot each of the above LOO analyses to see the Pareto shape k parameter for each point which gives us some indication of the pointwise predictive error for this data.

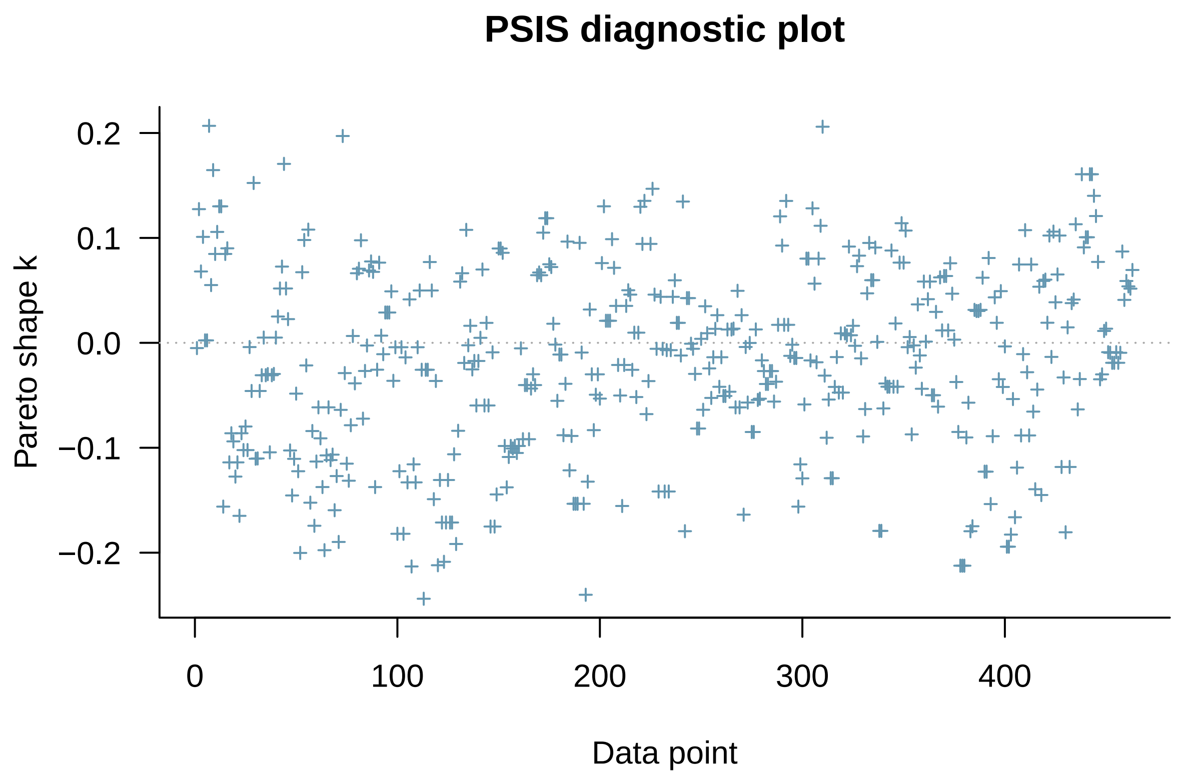
```
plot(lbf0)
```



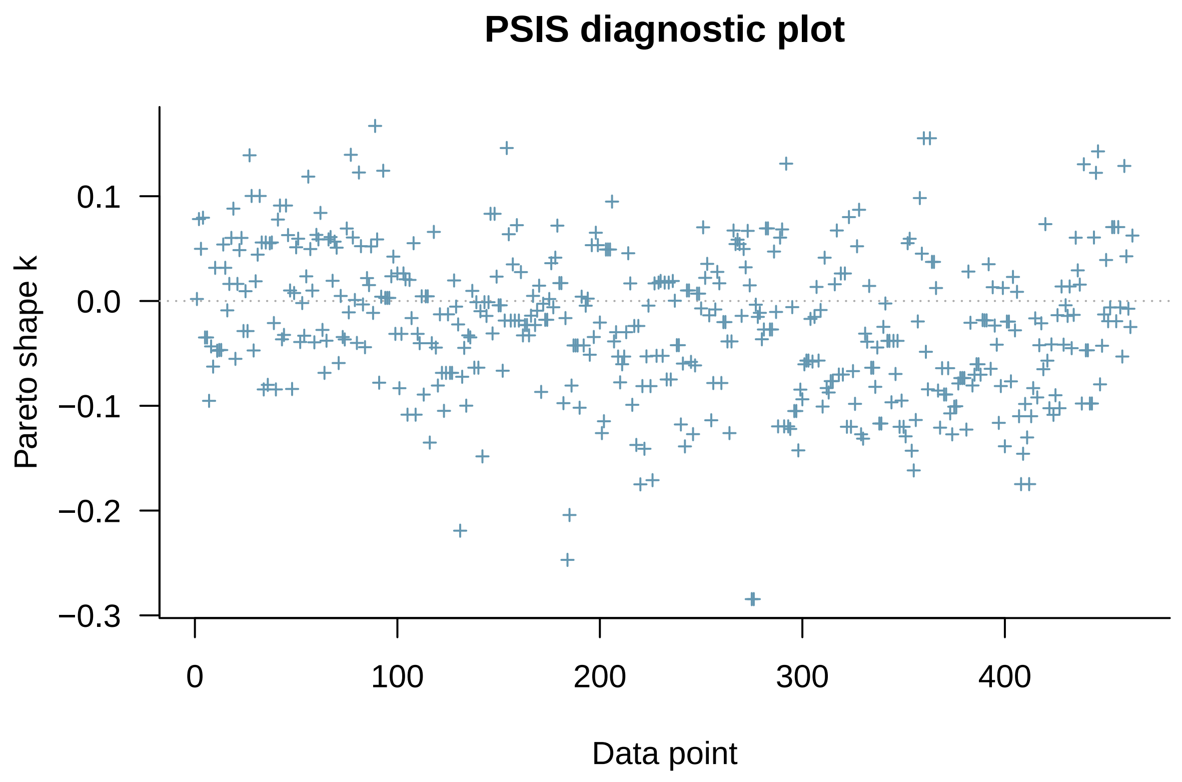
```
plot(lbf1)
```



```
plot(lbf2)
```


PSIS diagnostic plot
Pareto shape k
0.2
0.1
0.0
−0.1
−0.2
0
100
200
300
400
Data point

```
plot(lbf3)
```



With the above plots, we can notice a few general patterns. Across all 4 models, we can note that there is generally more predictive error at the two ends of the data—this is especially true for the left hand side of these graphs which corresponds to instructors with lower beauty scores. There are also a couple of data points (likely outliers of some kind) that have anomalous predictive error in all or most of the models. Noticeable instances of these include that one point near 300 that is consistently far below the 0 line and the point near 0 that is consistently far above the 0 horizontal line.

12.3: Scale of regression coefficients

In the table provided in the problem description, the coefficients for certain variables (distance to border, distance to capital, population, and GDP per capita) have much smaller absolute values than the other coefficients for other variables. This is due to the scale that these variables exist on in relation to the other variable scales included in the model.

In particular, two of the other variables are measured in percentages which means they necessarily vary on a scale of 0 to 100. The other variable with a more substantial coefficient is the "conflict before 2000" variable. I am not sure what exactly this variable tracks or how it is measured, but if it were the number of large-scale civil conflicts before 2000, it would be reasonable to assume that for most countries this number likely lands somewhere between 0 and 20.

The scales of GDP per capita, population, and distance values exist on scales that are much wider and with larger scales than the variables described above. For instance, population varies from very small nations like Greenland (with a population of around 57,000—about 1/3 the population of Eugene) to very large nations like China (with a population of around 1.4 billion—about 8,300 times the population of Eugene). GDP per capita and distance variables vary on similar scales across several magnitudes.

Variables that vary across several magnitudes will logically have smaller coefficients associated with them because every incremental or decremental change in that variable is associated with a smaller change proportional to the scale that that variable exists on. Therefore, the discrete changes that the coefficient for these variables measures (i.e. how the response variables changes with every increase of 1 in population) will be smaller and make the coefficient smaller in comparison to other coefficients in the model.

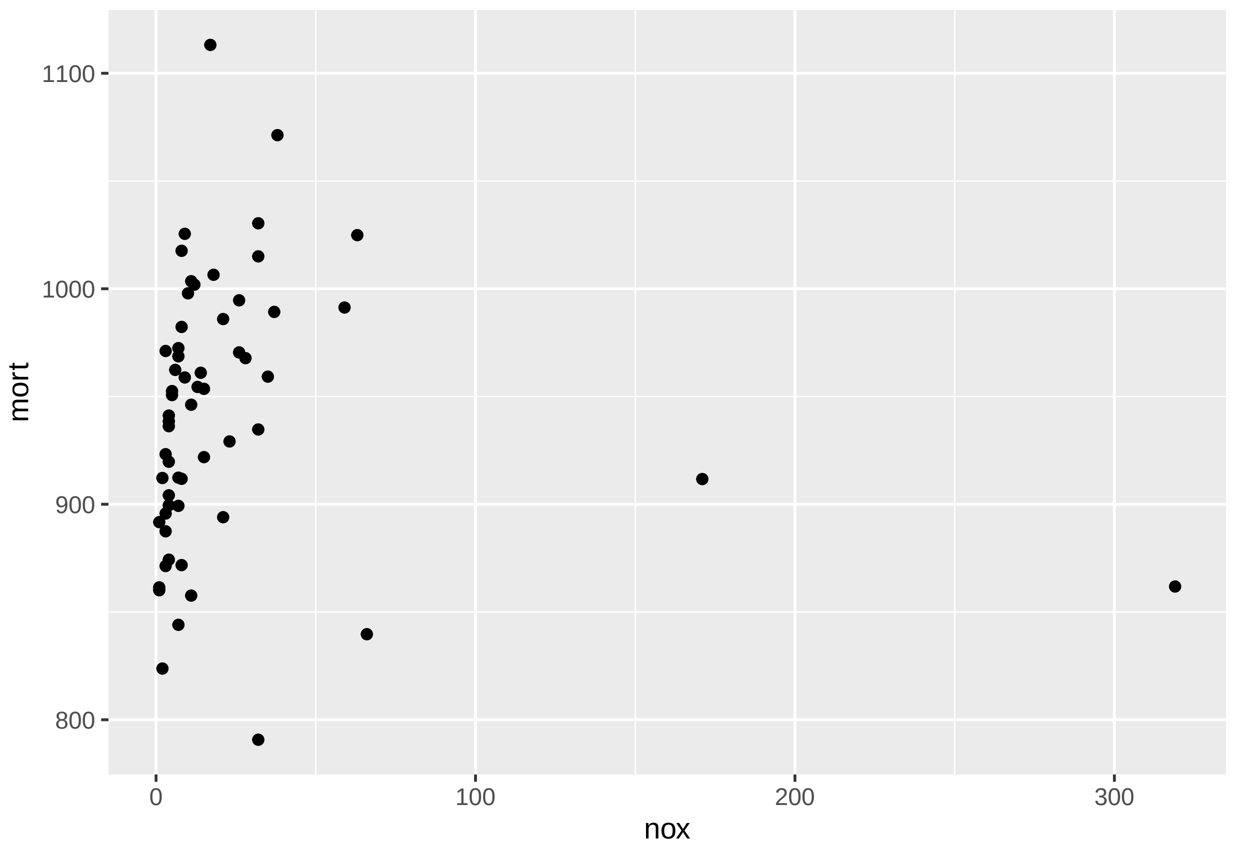
12.6: Logarithmic transformations

First we can import the pollution data from the textbook authors' GitHub site.

```r
myfile = "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/Pollution/data/pollution.csv"
pollution <- read.csv(myfile, header = T)
```

Now we can create a scatter plot of the data with nitric oxide levels as our x variable and mortality as our y variable.

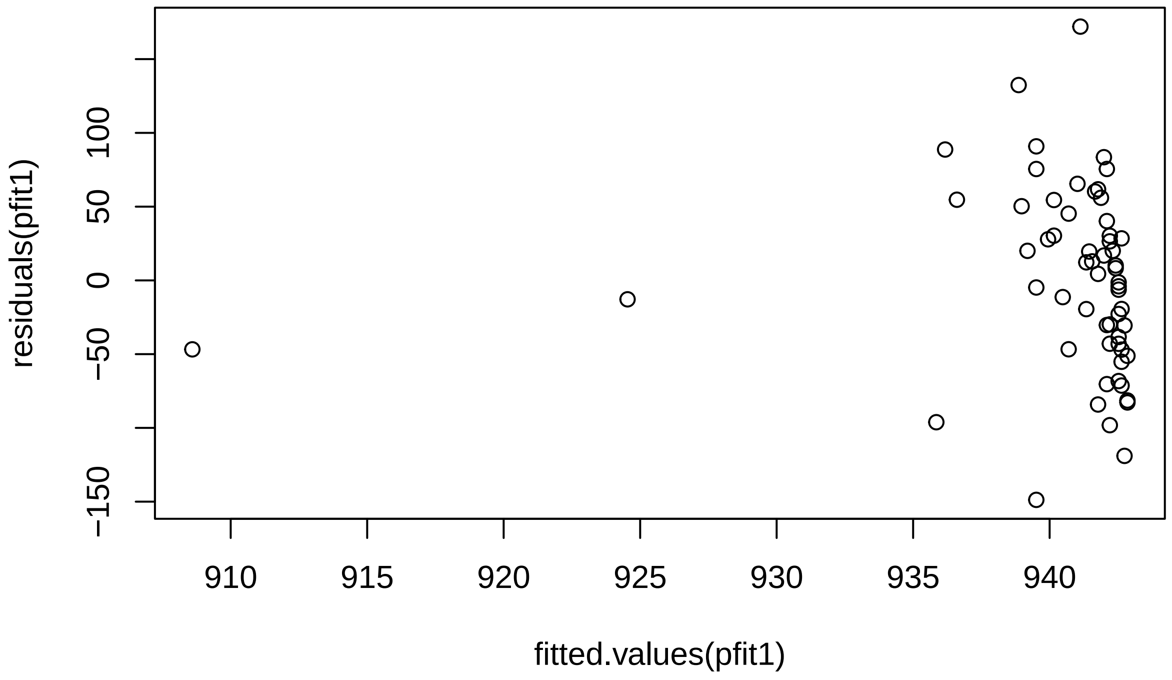
```r
ggplot(data=pollution, aes(x=nox, y=mort)) + geom_point()
```



From the data displayed graphically above, a linear regression model does not seem like it would be the most effective for modeling the interaction between the predictor nox and the response variable of mortality. The data from 0 to 50 could have a roughly linear relationship, but when we move beyond 50 on our nox scale, we end up with outliers that clearly do not have the same linear relationship as we could fit in the 0 to 50 range.

We can further illustrate the ineffectiveness of a linear model without transformations by fitting such a model and evaluating its fit. The below does just that and then plots a residual plot for the regression which will help us in seeing if the regression analysis meets the assumptions inherit in using such a model.

```r
pfit1 = stan_glm(mort~nox, data=pollution, refresh=0)
plot(residuals(pfit1)~fitted.values(pfit1), data=pollution)
```



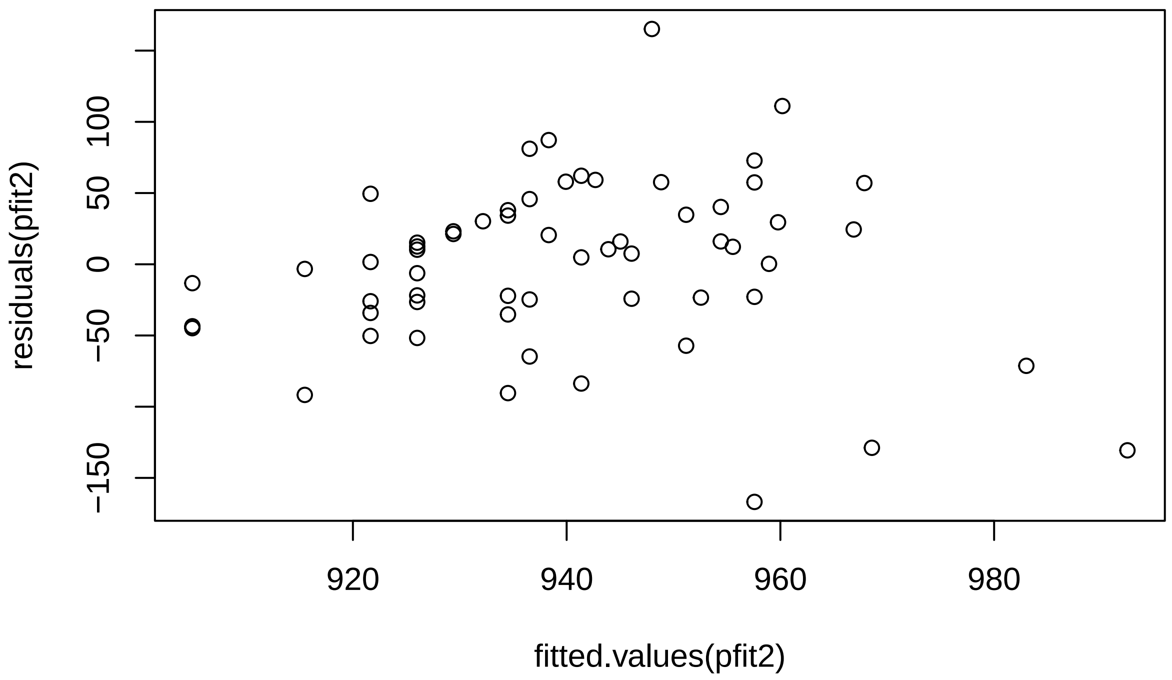
```r
print(pfit1)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      mort ~ nox
##  observations: 60
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) 943.0    9.0
## nox          -0.1    0.2
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 62.7    5.6
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

If a linear regression model is effective in modeling a relationship seen in data, then we expect the residual plot of the model to not display any strong patterns; this is not something we can say for the above residual plot. This residual plot essentially follows a pattern that is a mirrored version of the graphed data points, which means that the regression model could be compared to a horizontal (information-less) linear model. Furthermore, we can look beyond the residual plot, at the model itself and note that this model suggests a slightly negative correlation between nitric oxide pollution levels are mortality. We should be open to any conclusions we draw from the data, but stepping outside of this open-mindedness for a second we can recognize that a negative relationship between pollutants and mortality doesn't exactly make logical sense.

This trouble is likely due to the fact that the nox variable varies over several degrees of magnitude which means that our model may benefit from a logarithmic transformation on the predictor. We can make such a transformation with the R code below and then fit a new model to the data and check its residual plot in comparison to the plot we generated above.

```r
pfit2 = stan_glm(mort~log(nox), data=pollution, refresh=0)
plot(residuals(pfit2)~fitted.values(pfit2), data=pollution)
```



```r
print(pfit2)
```

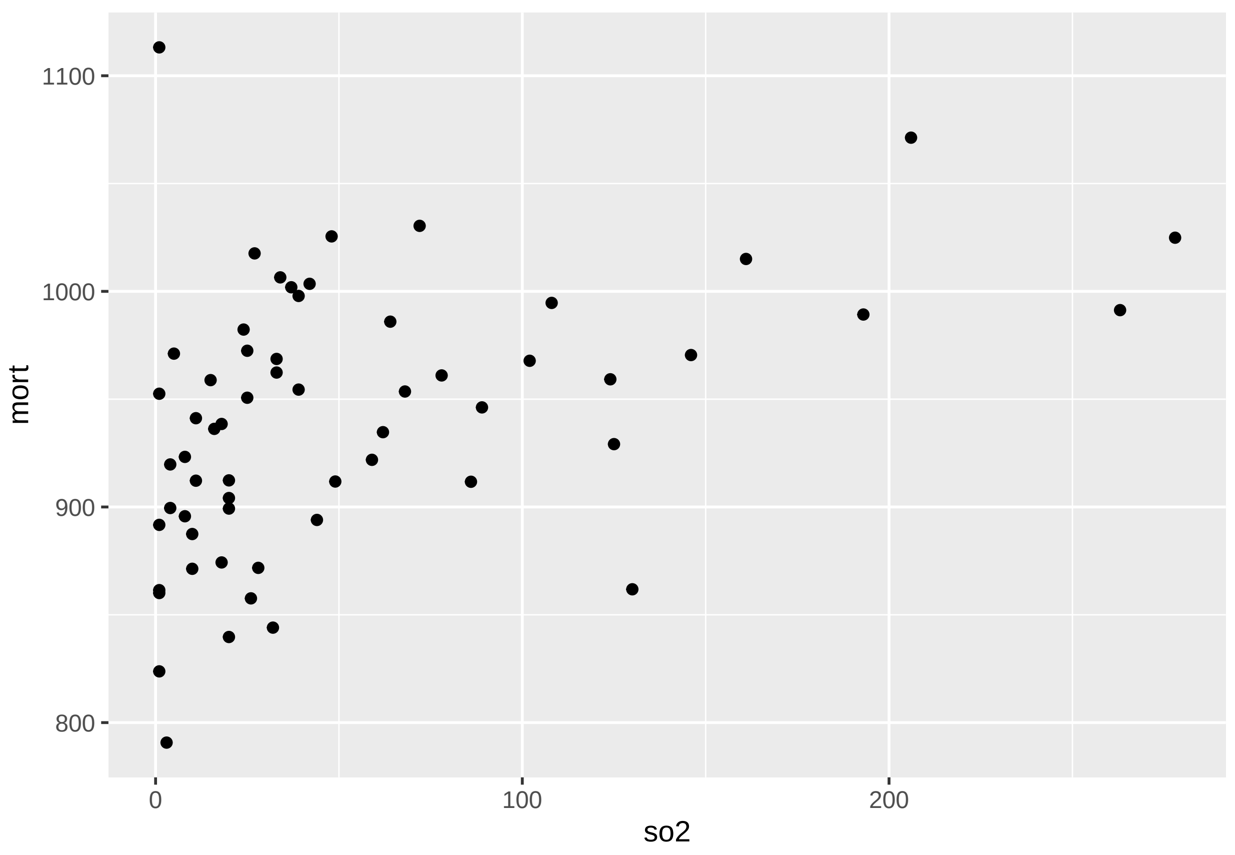
```
## stan_glm
##  family:       gaussian [identity]
##  formula:      mort ~ log(nox)
##  observations: 60
##  predictors:   2
## ------
##             Median MAD_SD
## (Intercept) 905.0   17.9
## log(nox)     15.2    6.9
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 60.2    5.5
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

The new residual plot is much closer to the type of residual plots we would expect from a effective linear regression model. Here, there are no obvious patterns in the points, nor any particular biases in either the fitted values or the residuals. Also, the residuals are roughly randomly distributed around 0 which further strengthens the argument that this residual plot is indicative of a better model than the previous plot.

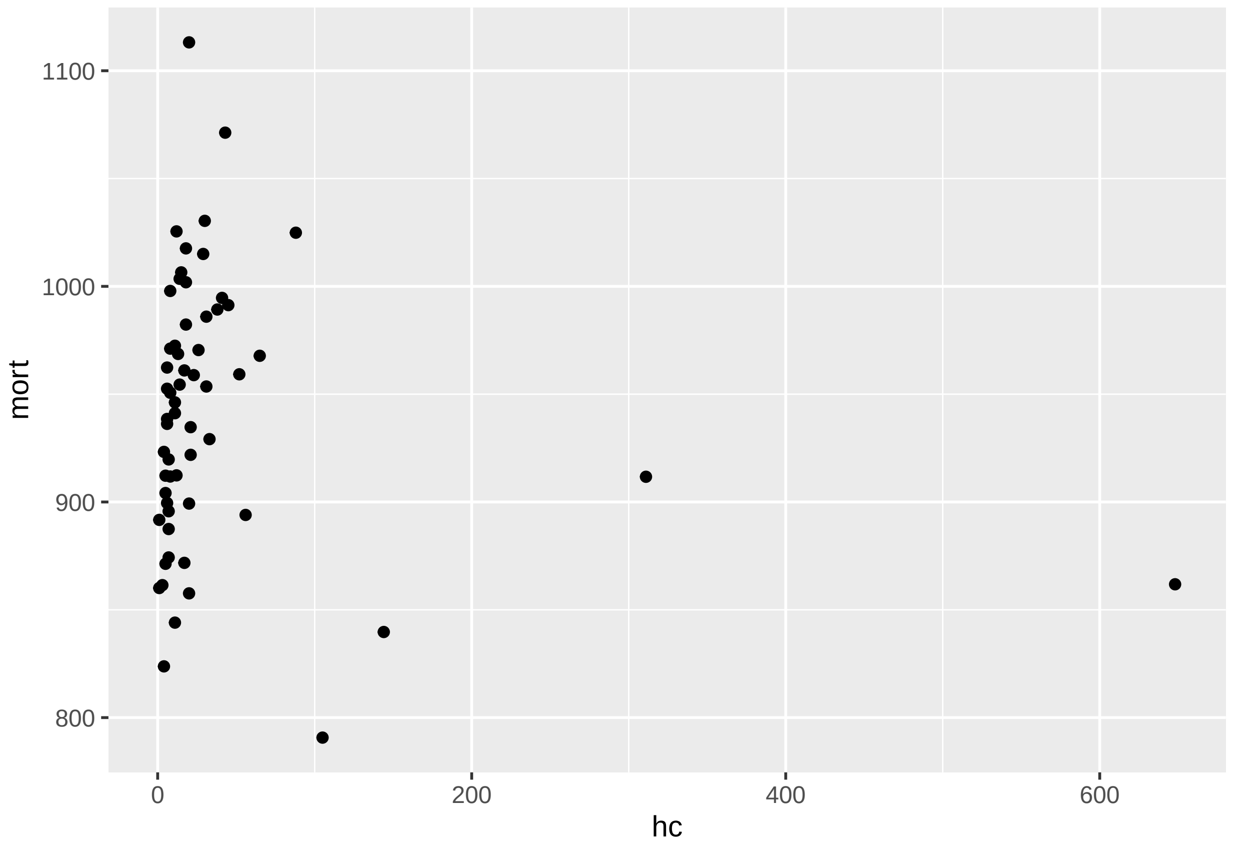
The slope coefficient generated from the model where we have applied a logarithmic transformation to the nox variable will be different every time because of pseudo random processes involved in Bayesian regression analysis. However, after running the above code several times, I have found that the coefficient's variation is generally centered around 15.3. This coefficient tells us that every e factor increase in nitric oxide levels is associated with an average increase of 15.3 in the mortality variable. From this coefficient we can conclude that this data suggests that higher nitric oxide levels are associated with higher rates of mortality.

We can now make full use of the data provided by including sulfur dioxide and hydrocarbons as predictors in addition to nitric oxide levels. We can consider what transformations might be reasonable for these other predictors by creating scatter plots similar to the ones we generated above relating the other pollutants to mortality.

```r
ggplot(data=pollution, aes(x=so2, y=mort)) + geom_point()
```



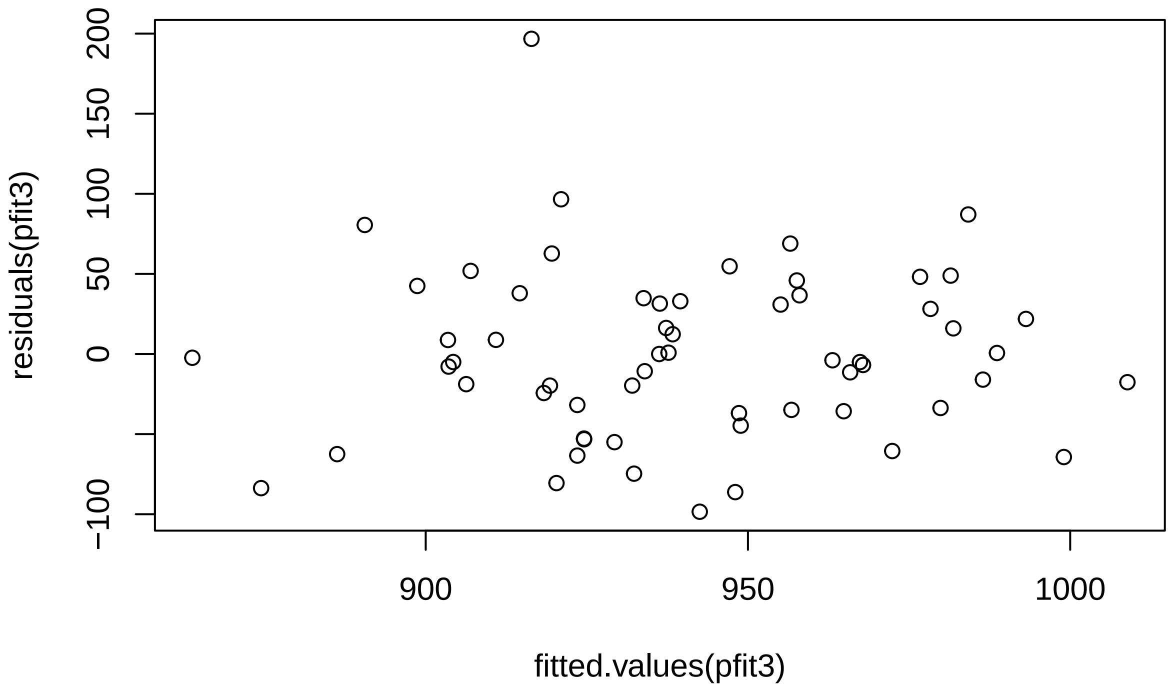
```
ggplot(data=pollution, aes(x=hc, y=mort)) + geom_point()
```



In the above scatter plots we see a similar distribution of the predictor variables as we saw before when plotting nitric oxides. Earlier, we improved our model by taking the log of the nox variable, so it stands to reason that doing a logarithmic transformation for these two new predictors could also be a good idea. Another justification for why we should take the logarithm of hc and so2 is that, like nox, the values are spread over several magnitudes, which further suggests a log transformation could help.

We can now make such transformations and fit a linear regression model with these 3 values as predictors using the R code below. In addition to generating this model, we will generate a residual plot to go along with it to evaluate if it's a valid model to use:

```
pfit3 = stan_glm(mort~log(nox) + log(hc) + log(so2), data=pollution, refresh=0)
plot(residuals(pfit3)~fitted.values(pfit3), data=pollution)
```



```
print(pfit3)
```

```
## stan_glm
##  family:       gaussian [identity]
##  formula:      mort ~ log(nox) + log(hc) + log(so2)
##  observations: 60
##  predictors:   4
## ------
##             Median MAD_SD
## (Intercept) 923.5   21.8
## log(nox)     55.0   22.0
## log(hc)     -54.4   19.3
## log(so2)     12.2    7.3
##
## Auxiliary parameter(s):
##       Median MAD_SD
## sigma 54.8    5.1
##
## ------
## * For help interpreting the printed output see ?print.stanreg
## * For info on the priors used see ?prior_summary.stanreg
```

The above residual plot does seem to satisfy the properties we would expect with an effective model; specifically, the fitted values and residuals do not exhibit any obvious bias, there is no evidence of any strong patterns, and the residuals appear to be roughly randomly distributed around the horizontal line representing a residual of 0.

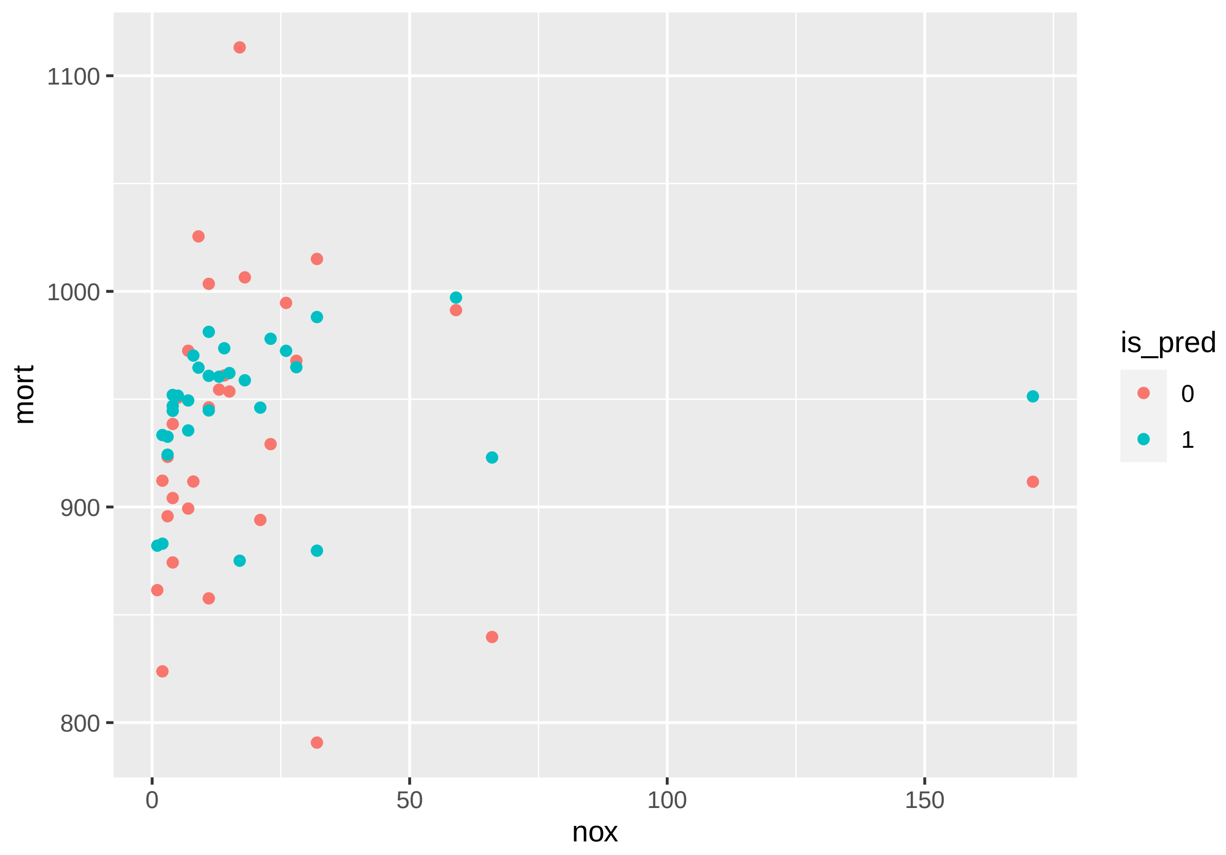
Given that we do not see any glaring red flags in our residual plot, we can turn to the fit we have generated and interpret what its coefficients are telling us. Like above, the coefficients relate the logarithm of the predictors to the response variable, so every e factor increase in one of these variables is associated with an increase of the respective coefficient's value in the mortality response. The important thing to note with this model is that because we have multiple predictors, these relationships between response and predictors assumes that all other predictors are set to constant values (we usually use their means to end up with the most accurate predictions).

Now that we have an effective model for mortality based on 3 pollutant predictors, we can generate an instance of this model using the first half of the data and then predict the values in the second half of the data. This will provide us with a rudimentary kind of cross validation—which will not be true cross validation because we used all the data to decide what model to use in the above steps.

```r
# Splitting up the data
half1_pollution = pollution[1:30,]
half2_pollution = pollution[31:60,]
# Making the model
pfit4 = stan_glm(mort~log(nox) + log(hc) + log(so2), data=half1_pollution, refresh=0)
# Predicting for the second half of the data
p_predict = predict(pfit4, newdata=half2_pollution)
```

We can then graphically display the predicted response values along with the training response variables to give us some idea of how our model worked out.

```r
copy = data.frame(half2_pollution)
copy$mort <- p_predict
copy$is_pred = as.factor(replicate(30,1))
half2_pollution$is_pred <- as.factor(replicate(30,0))
combined = rbind(half2_pollution, copy)
ggplot(data=combined, aes(x=nox, y=mort, color=is_pred)) + geom_point()
```



From this graphical representation, we can see that the predictions, although not perfect, do give us a pretty good idea of what the second half of the data may look like based on the model fitted to the first half.

12.7: Cross validation comparison of models with different transformations of outcomes

First we can write a little but of R code to retrieve the data from the internet (just like we have done in all the above problems).

```
myfile <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/Earnings/data/earnings.csv"
earnings = read.csv(myfile, header=T)
```

Now we can fit two models to this data: both will have height and sex as predictors for the response variable (earnings), but one will perform a logarithmic transformation on the variable earnk while the other will leave earnk as is. Note that for the log(earnk) model, we will have to add some small value to all the earnings values (here we chose 10) because some data points have an earnings of 0 and taking the log of zero leads to undefined behavior. We'll use a Bayesian approach to generate both models, which can be achieved with the code below

```
efit = stan_glm(earnk~height + male, data=earnings, refresh=0)
log_efit = stan_glm(log(earnk + 10)~height + male, data=earnings, refresh=0)
```

We now want to compare these two models by seeing how they perform with cross-validation, specifically leave-one-out (LOO) cross validation. We can conduct LOO analysis on each one of these fits individually, but we CANNOT directly compare their ELPD values because of how the nonlinear log transformation warps the continuous earnings variable in the second model.

However, we can adjust the loo analysis of the log_efit to be on a scale comparable to the loo analysis of efit by adding the Jacobian corresponding to nonlinear transformation we have made on the response variable. The Jacobian of log(y) is 1/y, so we want to add log(1/y) which is equivalent to subtracting log(y). The first step of doing this adjustment is to conduct LOO cross validation on both models without any adjustments, which we will do below:

```
eloo = loo(efit)
log_eloo = loo(log_efit)
```

We can then make the necessary adjustments to log_eloo to make a new variable log_eloo_with_J that represents a LOO analysis of the second fit with the Jacobian adjustment added in, allowing us to compare it to the first fit through cross validation analysis.

```
log_eloo_with_J <- log_eloo
log_eloo_with_J$pointwise[,1] <- log_eloo_with_J$pointwise[,1] - log(earnings$earnk + 10)
sum(log_eloo_with_J$pointwise[,1])
```

```
## [1] -7388.403
```

```
loo_compare(eloo, log_eloo_with_J)
```

```
##          elpd_diff se_diff
## log_efit    0.0       0.0
## efit     -766.4     153.8
```

Like with previous comparisons between LOO analyses generated for different models, we want the model that has the higher ELPD value, or in the case of the loo_compare function, the model that has an elpd_diff value of 0. We therefore conclude that the log_efit model where we take the log of our response variable is the one with a greater predictive ability according to this particular test. Furthermore, the standard error of the difference is small enough in comparison to the size of the difference where we are very unlikely to conclude that this difference is simply caused by random variation.

The problem now asks us to do this same procedure for some of the models we generated for problems earlier in the homework set. We can start out by following this procedure to compare the ozone models we made above, specifically ofit2 which has temperature and humidity as predictors for ozone levels and ofit5 which uses the same predictors and log of ozone levels as its response variable.

```
oloo = loo(ofit2)
log_oloo = loo(ofit5)
log_oloo_with_J <- log_oloo
log_oloo_with_J$pointwise[,1] <- log_oloo_with_J$pointwise[,1] - log(ozone$O3)
sum(log_eloo_with_J$pointwise[,1])
```

```
## [1] -7388.403
```

```
loo_compare(oloo, log_oloo_with_J)
```

```
##       elpd_diff se_diff
## ofit5   0.0       0.0
## ofit2 -39.3      13.4
```

This comparison also shows that the ELPD for the log-transformed data is higher (and higher by a statistically significant amount according to the standard error of the difference in ELPD estimates). This suggests that the log-transformed model has a higher predictive ability which is good to hear because that is the model we selected based on its residual plot as well.

There are no other data sets in this model where we generated both a model with the normal response variable and a model with the log-transformed response variable, but if there were, we could do analysis essentially identical to the two above examples to compare their predictive power.