Obsidian Data Analysis

Preliminary exploration and Data cleaning

To begin our analysis, we first import the dataset and conduct a preliminary examination of its variables.

```
obsidian = read.table(file = "/Users/minseoksong/Downloads/obsidian_data.txt", header=TRUE, sep= ",")
head(obsidian)
```

```
##             ID  mass  type     site element_Rb element_Sr element_Y element_Zr
## 1  288275.002a 0.502 Blade Ali Kosh        238         45        29        334
## 2 288275.002aa 0.227 Flake Ali Kosh        234         44        28        325
## 3 288275.002ab 0.188 Flake Ali Kosh        255         50        32        337
## 4 288275.002ac 0.153 Flake Ali Kosh        231         46        28        327
## 5 288275.002ad 0.102 Blade Ali Kosh        252         49        31        331
## 6 288275.002ae 0.440 Flake Ali Kosh        234         44        28        327
```

Upon initial inspection, it appears that the dataset contains some missing values. Let's investigate this further.

```
missing_ind = which(apply(is.na(obsidian),1,any) == TRUE)
missing_ind
```

```
## [1] 171 178
```

Rows 171 and 178 exhibit missing values. To enhance the precision of our regression analysis, we will temporarily exclude these rows rather than imputing them.

```
obsidian = obsidian[-missing_ind,]
```

This decision is justified by the ample size of our dataset, ensuring that the omission of a few rows will not significantly impact the overall robustness of our analysis.

```
dim(obsidian)[1]
```

```
## [1] 650
```

Another observation is the unusually high maximum value for mass. We suspect this could be a result of data corruption.

```
obsidian[which(obsidian$mass>30),]
```

```
##          ID mass  type         site element_Rb element_Sr element_Y element_Zr
## 465 297032q  160 Flake Chagha Sefid        214         41        27        312
```

Upon closer examination of this statistics, we observe that the quantities of four elements—Rb, Sr, Y, and Zr—do not exhibit significant deviations from the rest of the dataset. Given this consistency, it's reasonable to conclude that this data point may be corrupted. Consequently, we have opted to remove it from our analysis.

```
obsidian = obsidian[-which(obsidian$mass>30),]
```

Thirdly, upon again examining the dataset (details omitted here due to space constraints), we find that some of the "type" labels are ambiguously defined.

```
uncertaintypes = which(nchar(obsidian$type)>6)
length(uncertaintypes)
```

```
## [1] 34
```

While removing the 34 ambiguous data points might seem like a straightforward solution, such an action could introduce bias into our analysis. Instead, we aim to consolidate these labels based on our best interpretation. Specifically:

- Labels such as "Core fragment?", "Core fragment," "Cores and frags," and "Core/Fragment" can be uniformly categorized as "Core."
- The label "Flake (listed as)" can be simplified to "Flake."
- Both "Distal end of prismatic blade?" and "Retouched blades" can be classified under the "Blade" category.

By making these adjustments, we preserve the integrity of our dataset while clarifying ambiguous labels.

```
# Loop through each row of the obsidian data frame
for (i in 1:nrow(obsidian)){
  if (obsidian$type[i] == "Core fragment?"  || obsidian$type[i] == "Core fragment"  || obsidian$type[i]
    obsidian$type[i] = "Core"
  }
  else if (obsidian$type[i] == "Flake (listed as)" || obsidian$type[i] == "Used flake"){
    obsidian$type[i] = "Flake"
  }
  else if (obsidian$type[i] == "Distal end of prismatic blade?" || obsidian$type[i] == "Retouched blades
    obsidian$type[i] = "Blade"
  }
}
```

To ensure consistency and accuracy, we aim to standardize the capitalization and grammatical number (singular/plural) of nouns throughout the dataset:

- All labels will be converted to lowercase to maintain uniformity in capitalization.
- Singular forms of nouns will be used consistently to avoid discrepancies between singular and plural labels.

```
for (i in 1:dim(obsidian)[1]){
  i = as.double(i)
  if (obsidian$type[i] == "Blades" || obsidian$type[i] == "blade"){
    obsidian$type[i] = "Blade"
  }
  else if (obsidian$type[i] == "Flakes" || obsidian$type[i] == "flake"){
    obsidian$type[i] = "Flake"
  }
  else if (obsidian$type[i] == "core"){
    obsidian$type[i] = "Core"
  }
}
```

Additionally, we proceed to remove any remaining data points with ambiguous "type" covariates. This step ensures that our dataset is free from any uncertainties that could potentially skew our regression results.

```
obsidian = obsidian[-c(which(obsidian$type == "Core fragment? Flake?"), which(obsidian$type == "Blade/Fl
```

Upon examining the "site" covariate, we identify that the data point 215 has a label "Ali Kosh/Chaga Sefid," and data point 229 is labeled as "Hulailan Tepe Guran"—the only occurrence of this site in the dataset. Given that these are isolated instances and only represent two data points, we believe that removing them

will not introduce significant bias. Therefore, to maintain the clarity and consistency of our dataset, we opt to exclude these entries.

```
obsidian = obsidian[-which(nchar(obsidian$site)>13),]
```

With the data preprocessing steps completed, we are now ready to proceed with the regression analysis.

Model Selection

Before diving into the regression, we partition our dataset into a training set and a validation set. The training set will be employed for model selection, while the validation set will serve to mitigate multiple testing concerns and selective inference issues.

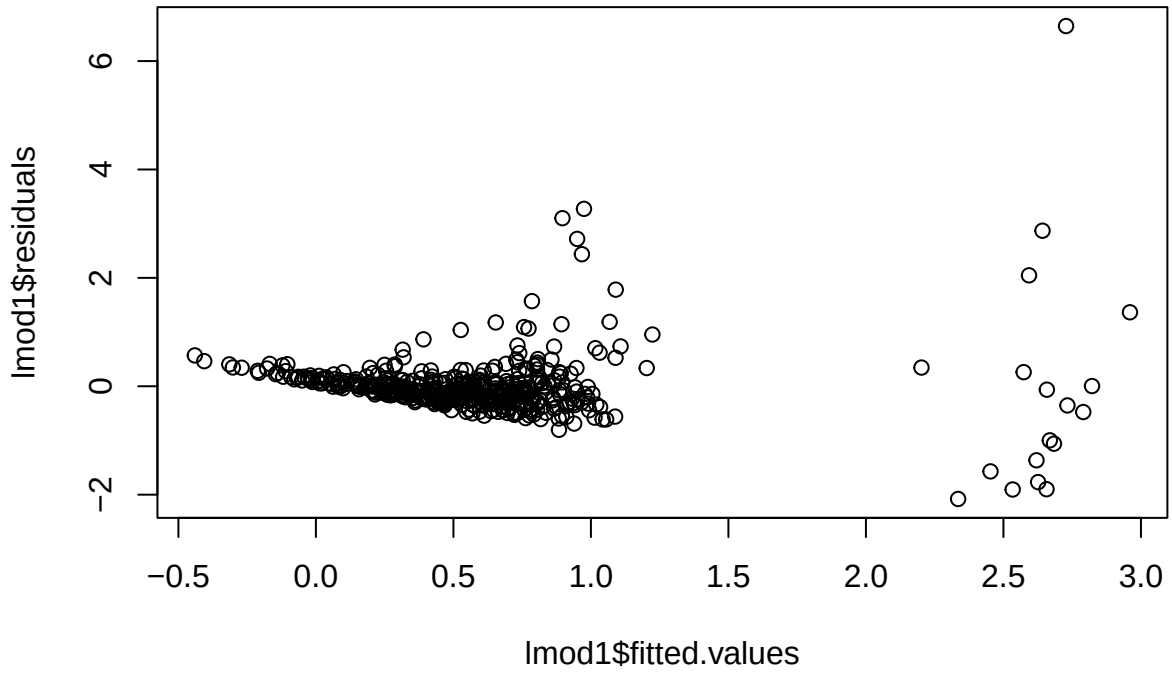
```
set.seed(1234)
n=dim(obsidian)[1]
ntrain = 410; nval = 203
validation = sample(n, nval, replace=FALSE)
train = setdiff(1:n, validation)
```

Initially, we opt to exclude the ID as a covariate in our regression model. Beyond the evident reason that each data point possesses a unique ID, there's an absence of discernible patterns that would justify its inclusion. For instance, when we limit the ID to its first three digits, it becomes perfectly collinear with the "site" covariate. Expanding the ID beyond these initial digits results in an overly complex model, which would inflate the variance excessively.

```
obsidian$type = as.factor(obsidian$type)
obsidian$site = as.factor(obsidian$site)
lmod1 <- lm(mass ~ type + site + element_Rb + element_Sr + element_Y + element_Zr, obsidian[train,])
summary(lmod1)
```

```
##
## Call:
## lm(formula = mass ~ type + site + element_Rb + element_Sr + element_Y +
##     element_Zr, data = obsidian[train, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0792 -0.2308 -0.0758  0.1152  6.6463
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       4.619833   0.844264   5.472 7.59e-08 ***
## typeCore          1.794495   0.152564  11.762  < 2e-16 ***
## typeFlake         0.103903   0.062862   1.653   0.0991 .
## siteChagha Sefid  0.062220   0.064331   0.967   0.3340
## element_Rb       -0.020048   0.003855  -5.200 3.09e-07 ***
## element_Sr       -0.009608   0.016731  -0.574   0.5661
## element_Y        -0.003951   0.014827  -0.266   0.7900
## element_Zr        0.003756   0.002988   1.257   0.2094
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6065 on 428 degrees of freedom
## Multiple R-squared:  0.4268, Adjusted R-squared:  0.4174
## F-statistic: 45.52 on 7 and 428 DF,  p-value: < 2.2e-16
```

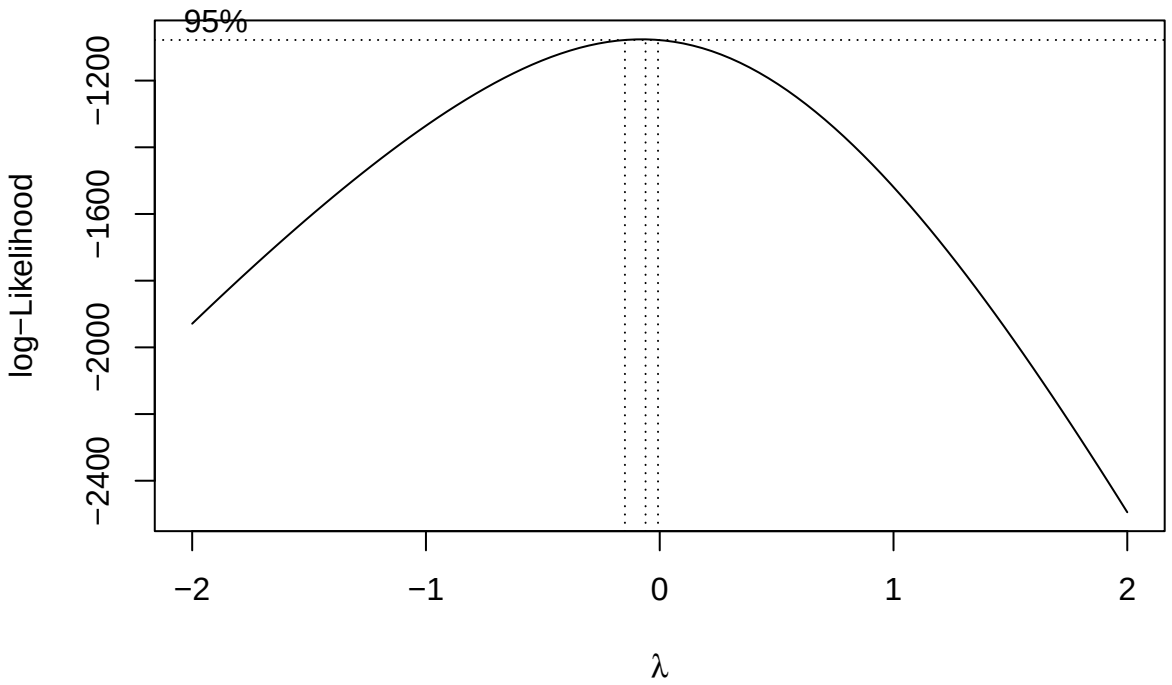
```r
plot(lmod1$fitted.values, lmod1$residuals)
```



Upon examination, we detect issues of nonlinearity and nonconstant variance concurrently. Given these challenges, the use of weighted least squares might not be the most appropriate approach. Instead, we are first inclined to consider a transformation of the data to address these concerns more effectively.

we will explore two potential transformations: the log transformation and the Box-Cox transformation. By comparing these methods, we aim to determine the most suitable approach to linearize our data and stabilize its variance.

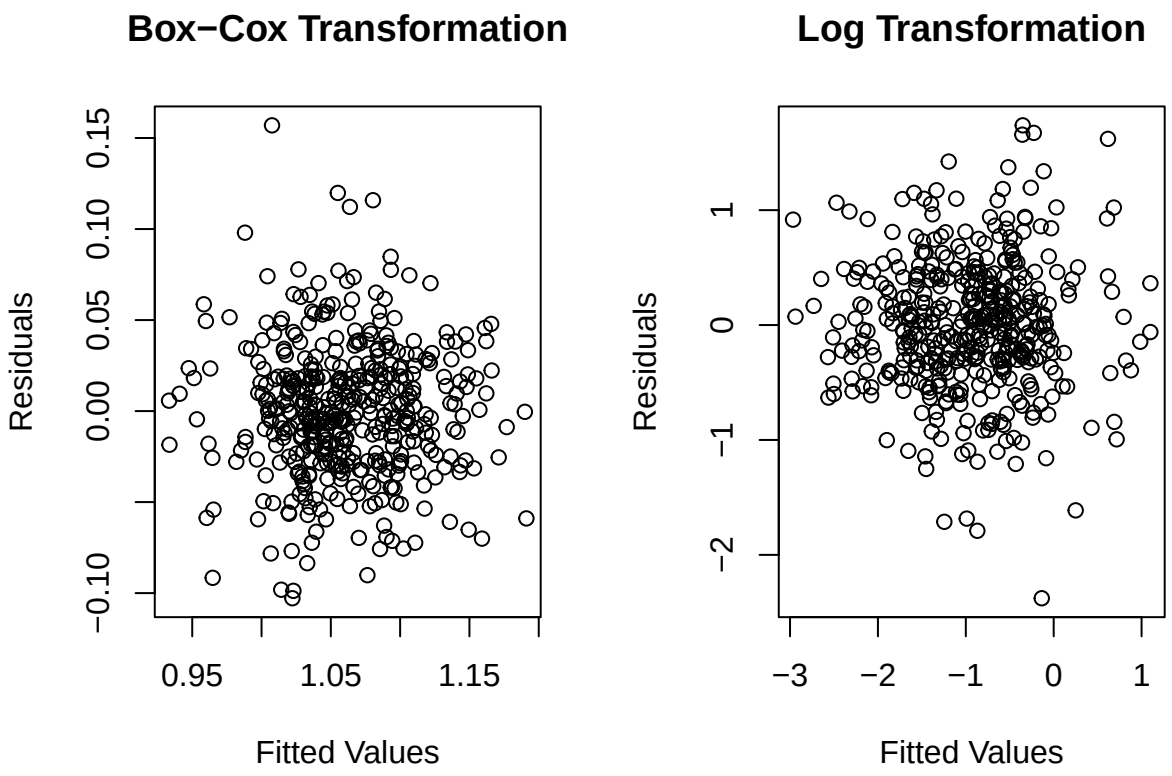
```r
library(MASS)
bc <- boxcox(lmod1)
```




```
bc$x[which.max(bc$y)]
```

```
## [1] -0.06060606
```

```
par(mfrow=c(1,2))

# Box-Cox Transformation
lmod1_boxcox = lm(mass^(-0.0606) ~ type + site + element_Rb + element_Sr + element_Y + element_Zr, obsi
plot(lmod1_boxcox$fitted.values, lmod1_boxcox$residuals, main="Box-Cox Transformation", xlab="Fitted Va

# Log Transformation
lmod1_log = lm(log(mass) ~ type + site + element_Rb + element_Sr + element_Y + element_Zr, obsidian[tra
plot(lmod1_log$fitted.values, lmod1_log$residuals, main="Log Transformation", xlab="Fitted Values", yla
```



Upon comparison, both the log transformation and the Box-Cox transformation effectively address our concerns regarding linearity and constant variance. Given their comparable performance, we opt for the log transformation, as it offers a more intuitive and natural interpretation for our dataset.

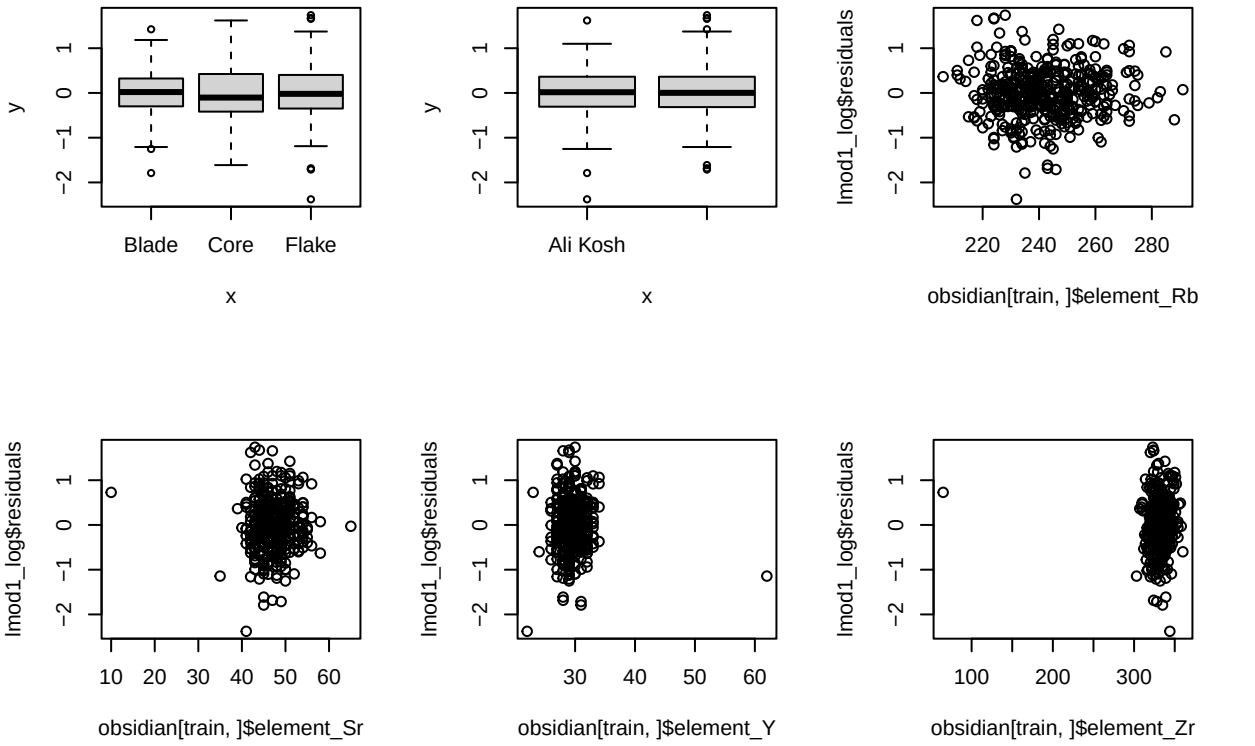
```
summary(lmod1_log)
```

```
##
## Call:
## lm(formula = log(mass) ~ type + site + element_Rb + element_Sr +
##     element_Y + element_Zr, data = obsidian[train, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.3775 -0.3099  0.0062  0.3623  1.7380
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)       5.754779   0.798390   7.208 2.59e-12 ***
## typeCore          0.899528   0.144275   6.235 1.09e-09 ***
## typeFlake        -0.018618   0.059447  -0.313 0.754287
## siteChagha Sefid  0.327625   0.060835   5.385 1.19e-07 ***
## element_Rb       -0.033275   0.003646  -9.127  < 2e-16 ***
## element_Sr       -0.054956   0.015822  -3.473 0.000566 ***
## element_Y        -0.033914   0.014021  -2.419 0.015988 *
## element_Zr        0.014091   0.002826   4.987 8.93e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5735 on 428 degrees of freedom
## Multiple R-squared:  0.6142, Adjusted R-squared:  0.6079
```

```
## F-statistic: 97.33 on 7 and 428 DF,  p-value: < 2.2e-16
```

Note that the intercept term implicitly treats the "Blade" type as a reference level. Therefore, even if the t-value for the "Flake" type covariate is high, we should refrain from excluding it. Given these considerations, our model appears to be valid. Let's proceed to validate it using the validation dataset.

```
par(mfrow=c(2,3))
plot(obsidian[train,]$type, lmod1_log$residuals)
plot(obsidian[train,]$site, lmod1_log$residuals)
plot(obsidian[train,]$element_Rb, lmod1_log$residuals)
plot(obsidian[train,]$element_Sr, lmod1_log$residuals)
plot(obsidian[train,]$element_Y, lmod1_log$residuals)
plot(obsidian[train,]$element_Zr, lmod1_log$residuals)
```



While we do observe some outliers in the data, the overall model appears to adhere well to the assumptions of linearity and constant variance. This suggests that our chosen model is robust and suitable for our dataset.

```
BIC(lmod1_log)
```

```
## [1] 799.1439
```

Given the structure of our model, it's pertinent to inquire about potential interaction terms. With $\binom{6}{2} = 15$ possible two-way interactions to explore, we need to adjust for multiple testing. Consequently, we'll employ a significance threshold of $\frac{0.05}{15} = 0.003$ to evaluate these interactions.

```
# Define the base variables
base_vars <- c("type", "site", "element_Rb", "element_Sr", "element_Y", "element_Zr")

# Define the interaction terms to test
interaction_terms <- expand.grid(base_vars, base_vars)
interaction_terms <- interaction_terms[interaction_terms$Var1 != interaction_terms$Var2, ]

# Compute BIC for each interaction term
```

```r
bic_values <- list()

for (i in 1:nrow(interaction_terms)) {
    formula_str <- paste("log(mass) ~ (", paste(base_vars, collapse=" + "), ") +",
                         interaction_terms$Var1[i], "*", interaction_terms$Var2[i])
    formula_obj <- as.formula(formula_str)
    model <- lm(formula_obj, data = obsidian[train, ])
    bic_values[[paste(interaction_terms$Var1[i], interaction_terms$Var2[i], sep = "*")]] <- BIC(model)
}

# Print the BIC values
bic_values
```

```
## $`site*type`
## [1] 803.2455
##
## $`element_Rb*type`
## [1] 808.7675
##
## $`element_Sr*type`
## [1] 811.1033
##
## $`element_Y*type`
## [1] 803.1618
##
## $`element_Zr*type`
## [1] 807.427
##
## $`type*site`
## [1] 803.2455
##
## $`element_Rb*site`
## [1] 805.2066
##
## $`element_Sr*site`
## [1] 804.9216
##
## $`element_Y*site`
## [1] 798.3041
##
## $`element_Zr*site`
## [1] 804.2896
##
## $`type*element_Rb`
## [1] 808.7675
##
## $`site*element_Rb`
## [1] 805.2066
##
## $`element_Sr*element_Rb`
## [1] 799.8755
##
## $`element_Y*element_Rb`
## [1] 792.7635
```

```
##
## $`element_Zr*element_Rb`
## [1] 783.486
##
## $`type*element_Sr`
## [1] 811.1033
##
## $`site*element_Sr`
## [1] 804.9216
##
## $`element_Rb*element_Sr`
## [1] 799.8755
##
## $`element_Y*element_Sr`
## [1] 786.9142
##
## $`element_Zr*element_Sr`
## [1] 790.2093
##
## $`type*element_Y`
## [1] 803.1618
##
## $`site*element_Y`
## [1] 798.3041
##
## $`element_Rb*element_Y`
## [1] 792.7635
##
## $`element_Sr*element_Y`
## [1] 786.9142
##
## $`element_Zr*element_Y`
## [1] 770.7243
##
## $`type*element_Zr`
## [1] 807.427
##
## $`site*element_Zr`
## [1] 804.2896
##
## $`element_Rb*element_Zr`
## [1] 783.486
##
## $`element_Sr*element_Zr`
## [1] 790.2093
##
## $`element_Y*element_Zr`
## [1] 770.7243
```

Given the results, we add the interaction term element_Y * element_Zr.

```
# Add the interaction term element_Y * element_Zr and recompute BIC
base_vars <- c(base_vars, "element_Y:element_Zr")
bic_values_with_interaction <- list()
```

```r
for (i in 1:nrow(interaction_terms)) {
    formula_str <- paste("log(mass) ~ (", paste(base_vars, collapse=" + "), ")")
    formula_obj <- as.formula(formula_str)
    model <- lm(formula_obj, data = obsidian[train, ])
    bic_values_with_interaction[[paste(interaction_terms$Var1[i], interaction_terms$Var2[i], sep = "*")]
}

# Print the BIC values with the added interaction
bic_values_with_interaction
```

```
## $`site*type`
## [1] 770.7243
##
## $`element_Rb*type`
## [1] 770.7243
##
## $`element_Sr*type`
## [1] 770.7243
##
## $`element_Y*type`
## [1] 770.7243
##
## $`element_Zr*type`
## [1] 770.7243
##
## $`type*site`
## [1] 770.7243
##
## $`element_Rb*site`
## [1] 770.7243
##
## $`element_Sr*site`
## [1] 770.7243
##
## $`element_Y*site`
## [1] 770.7243
##
## $`element_Zr*site`
## [1] 770.7243
##
## $`type*element_Rb`
## [1] 770.7243
##
## $`site*element_Rb`
## [1] 770.7243
##
## $`element_Sr*element_Rb`
## [1] 770.7243
##
## $`element_Y*element_Rb`
## [1] 770.7243
##
## $`element_Zr*element_Rb`
## [1] 770.7243
```

```
##
## $`type*element_Sr`
## [1] 770.7243
##
## $`site*element_Sr`
## [1] 770.7243
##
## $`element_Rb*element_Sr`
## [1] 770.7243
##
## $`element_Y*element_Sr`
## [1] 770.7243
##
## $`element_Zr*element_Sr`
## [1] 770.7243
##
## $`type*element_Y`
## [1] 770.7243
##
## $`site*element_Y`
## [1] 770.7243
##
## $`element_Rb*element_Y`
## [1] 770.7243
##
## $`element_Sr*element_Y`
## [1] 770.7243
##
## $`element_Zr*element_Y`
## [1] 770.7243
##
## $`type*element_Zr`
## [1] 770.7243
##
## $`site*element_Zr`
## [1] 770.7243
##
## $`element_Rb*element_Zr`
## [1] 770.7243
##
## $`element_Sr*element_Zr`
## [1] 770.7243
##
## $`element_Y*element_Zr`
## [1] 770.7243
```

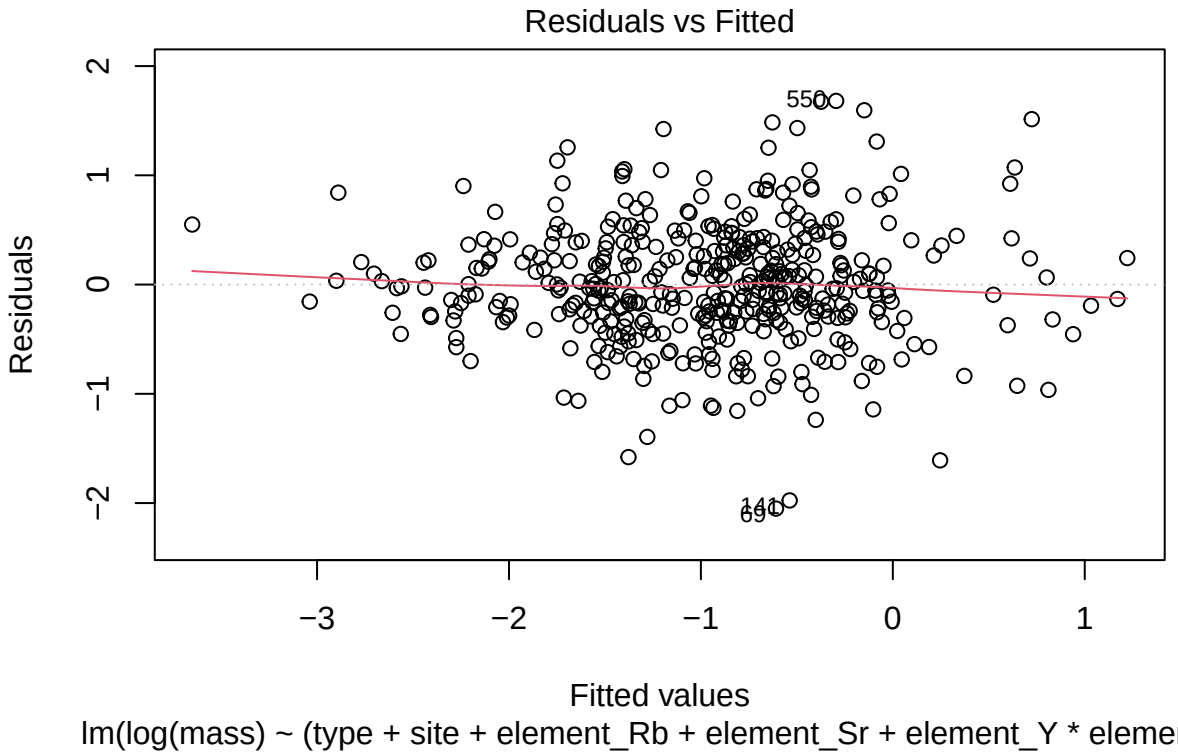
Based on our evaluations, all computed BIC values exceed 770. Consequently, we decide not to introduce any additional interaction terms to the model, as they do not provide a significant improvement in model fit.

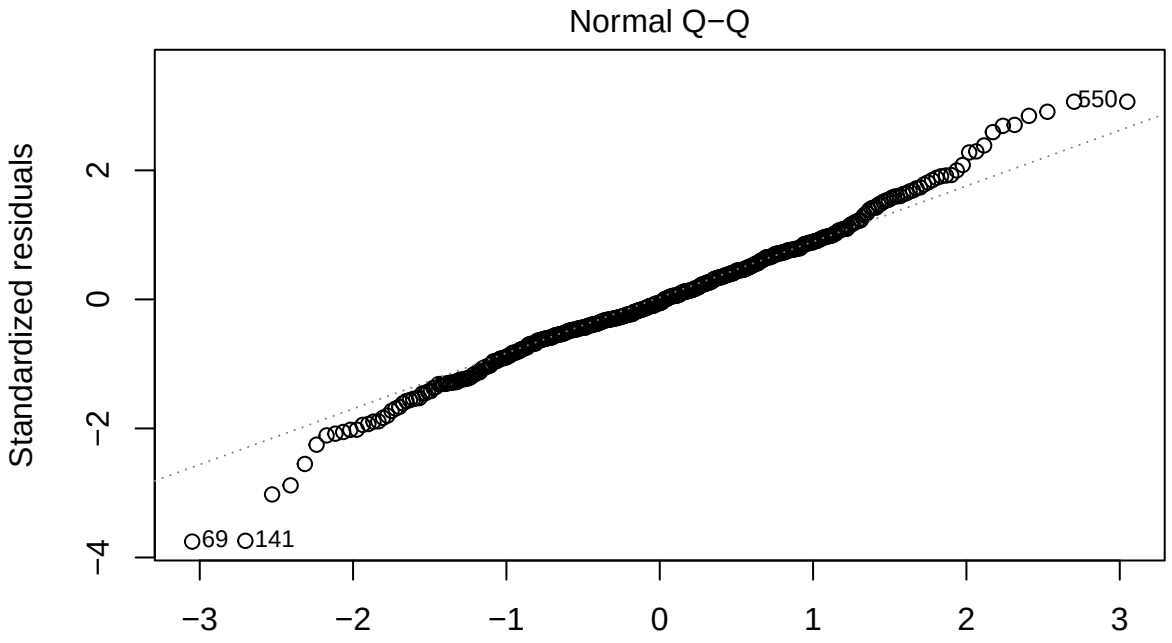
```
lmod2 <- lm(log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element_Zr), data = obsidia
summary(lmod2)
```

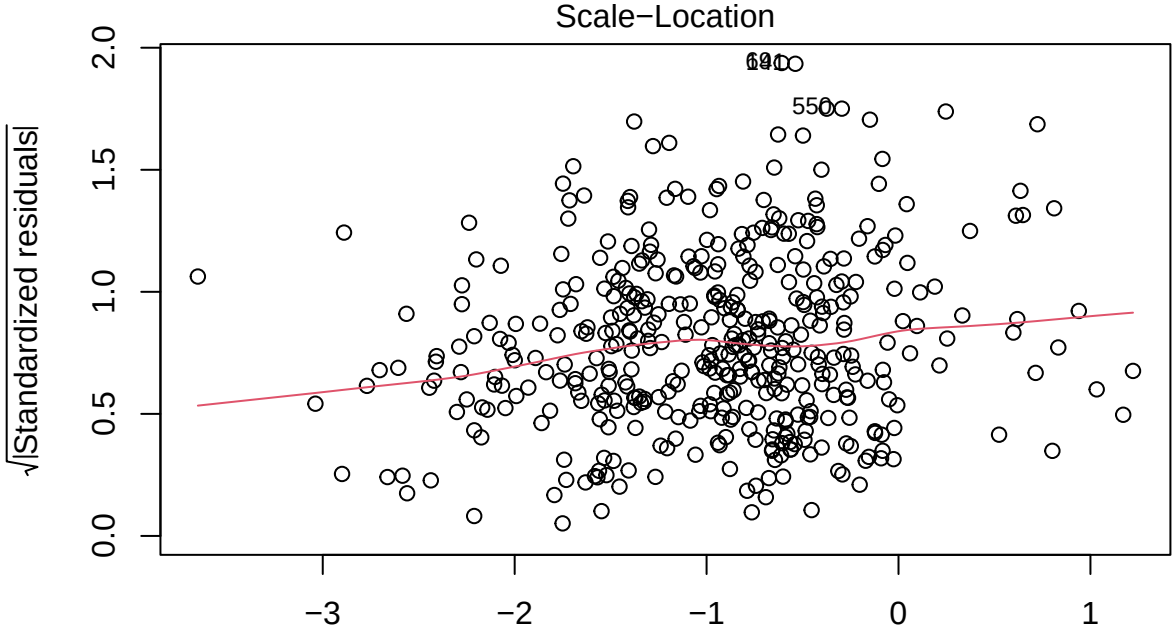
```
##
## Call:
## lm(formula = log(mass) ~ (type + site + element_Rb + element_Sr +
```

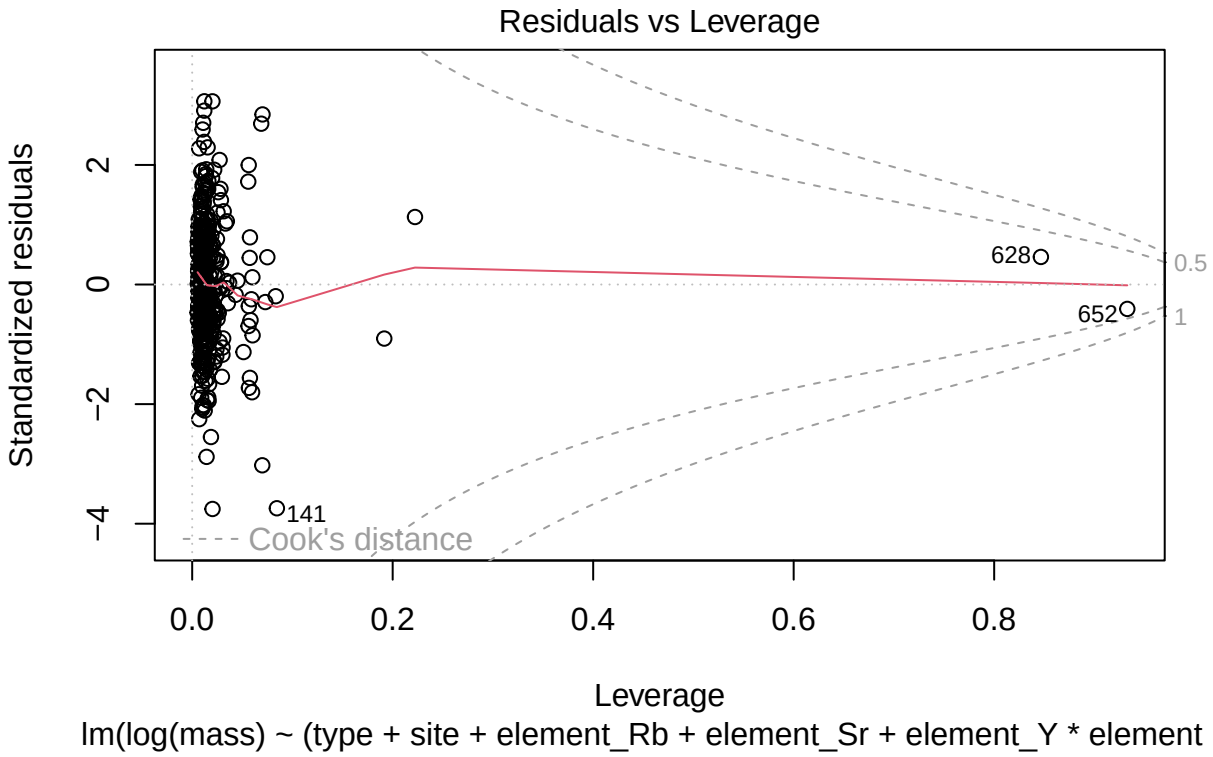
```
##     element_Y * element_Zr), data = obsidian[train, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.05050 -0.30275 -0.03086  0.33680  1.68102
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)          34.1563958  4.8511996   7.041 7.68e-12 ***
## typeCore              0.9024303  0.1388416   6.500 2.25e-10 ***
## typeFlake            -0.0279413  0.0572293  -0.488    0.626
## siteChagha Sefid      0.3532860  0.0587037   6.018 3.80e-09 ***
## element_Rb           -0.0476510  0.0042648 -11.173  < 2e-16 ***
## element_Sr           -0.0719382  0.0154933  -4.643 4.57e-06 ***
## element_Y            -1.0911642  0.1788165  -6.102 2.35e-09 ***
## element_Zr           -0.0649944  0.0136121  -4.775 2.48e-06 ***
## element_Y:element_Zr  0.0033962  0.0005728   5.929 6.28e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5519 on 427 degrees of freedom
## Multiple R-squared:  0.6435, Adjusted R-squared:  0.6369
## F-statistic: 96.36 on 8 and 427 DF,  p-value: < 2.2e-16
```

```
plot(lmod2)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element ...
550
69
141


Scale−Location
√|Standardized residuals|
Fitted values
lm(log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element ...
550



Upon visual inspection of the diagnostic plots, we observe slight nonconstant variance trends at both ends and a hint of nonlinearity. However, these trends are not pronounced enough to be of major concern.

To optimize our model selection, we'll employ the AIC criterion and utilize the step function for model comparison:

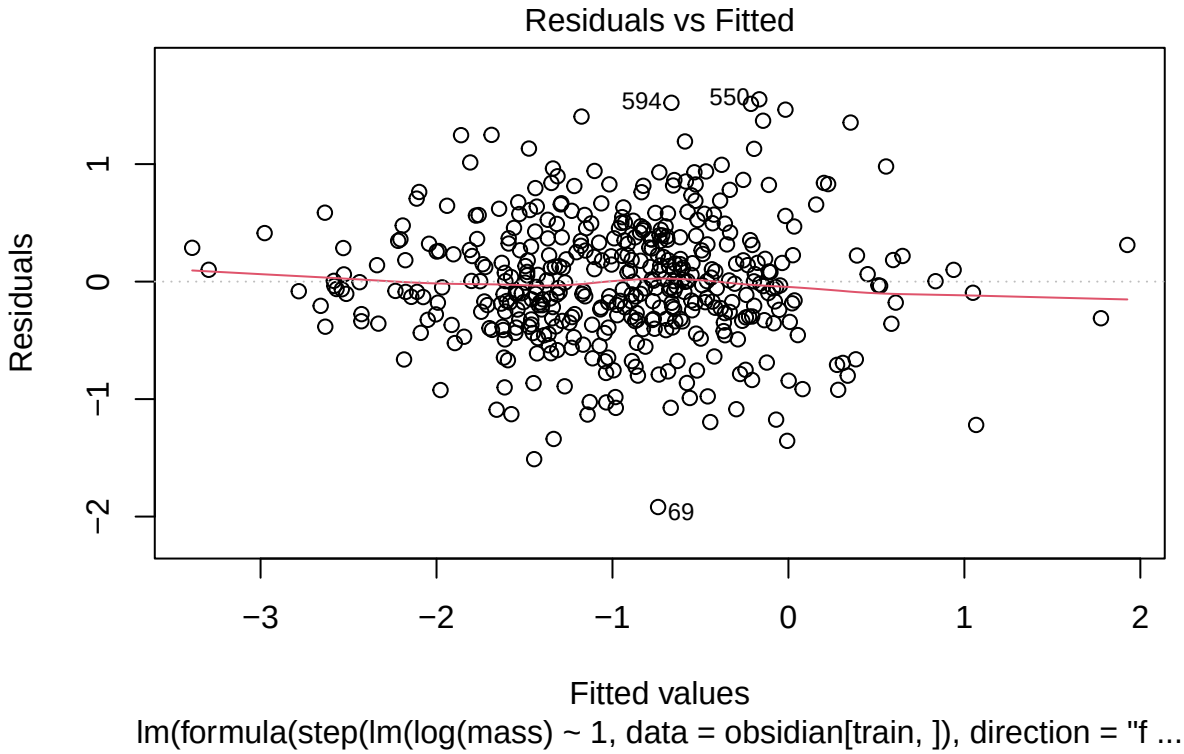
```
lmod3 <- lm(log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element_Zr)**2, data = obsi
lmod4 <- lm(formula(step(lm(log(mass)~1, data = obsidian[train, ]), direction='forward', scope=formula(l
summary(lmod4)
```

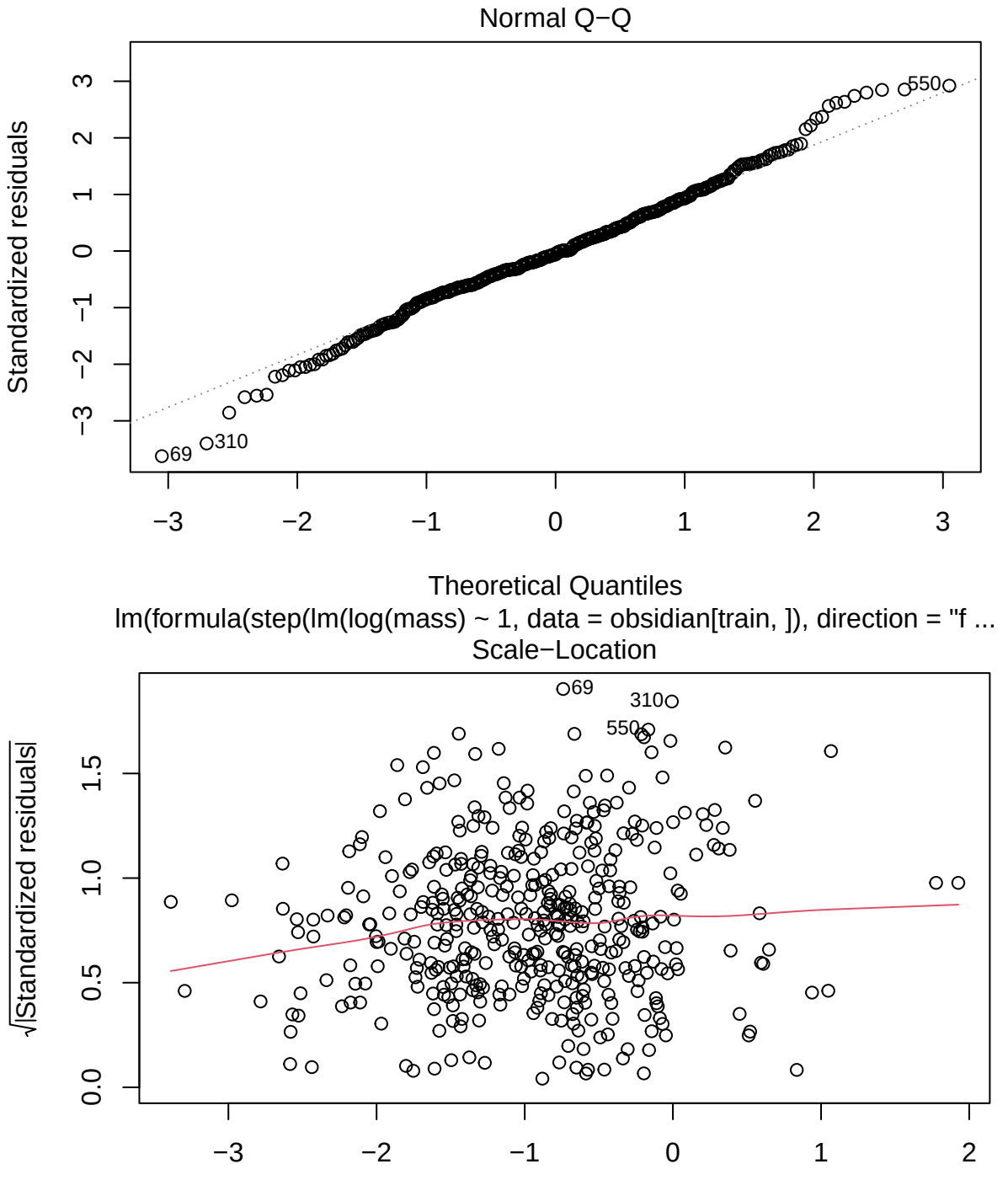
```
##
## Call:
## lm(formula = formula(step(lm(log(mass) ~ 1, data = obsidian[train,
##     ]), direction = "forward", scope = formula(lmod3), trace = 0)),
##     data = obsidian[train, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.91858 -0.32215 -0.03133  0.32270  1.55136
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)                76.5214410 16.7664473   4.564 6.61e-06 ***
## element_Rb                 -0.2146937  0.0889675  -2.413 0.016245 *
## typeCore                   -4.8445000  4.1872857  -1.157 0.247953
## typeFlake                  -0.0108872  1.0684526  -0.010 0.991875
## siteChagha Sefid            0.3812809  0.0742913   5.132 4.40e-07 ***
## element_Zr                 -0.2382416  0.0697217  -3.417 0.000695 ***
## element_Sr                  0.4218074  0.2610029   1.616 0.106828
## element_Y                  -1.4497058  0.6297122  -2.302 0.021817 *
## element_Rb:element_Zr       0.0004648  0.0003863   1.203 0.229544
```

```
## element_Rb:element_Sr       0.0003785  0.0010665   0.355 0.722840
## typeCore:siteChagha Sefid  -0.7960890  0.5316803  -1.497 0.135070
## typeFlake:siteChagha Sefid  0.0173943  0.1246439   0.140 0.889082
## element_Zr:element_Y        0.0063046  0.0028439   2.217 0.027170 *
## typeCore:element_Y          0.3675345  0.1733637   2.120 0.034594 *
## typeFlake:element_Y         0.1029781  0.0524898   1.962 0.050443 .
## element_Rb:typeCore        -0.0167357  0.0237771  -0.704 0.481915
## element_Rb:typeFlake       -0.0127106  0.0062643  -2.029 0.043089 *
## element_Sr:element_Y       -0.0145727  0.0073622  -1.979 0.048428 *
## element_Zr:element_Sr      -0.0004496  0.0003136  -1.434 0.152436
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.541 on 417 degrees of freedom
## Multiple R-squared:  0.6655, Adjusted R-squared:  0.6511
## F-statistic: 46.09 on 18 and 417 DF,  p-value: < 2.2e-16
```

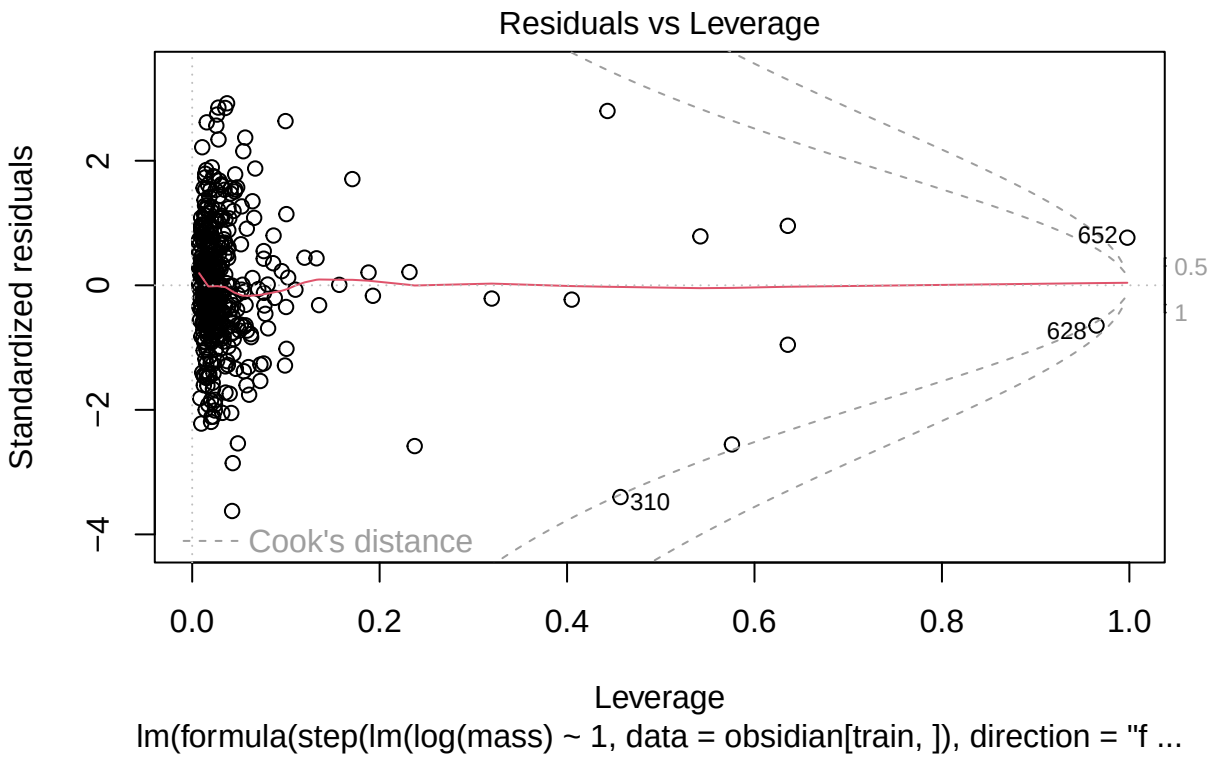
As anticipated, the model selected based on the AIC criterion is more lenient compared to the BIC, resulting in a larger model than lmod2. This is consistent with the general understanding that AIC tends to favor more complex models compared to BIC, which penalizes model complexity more heavily.

```
plot(lmod4)
```



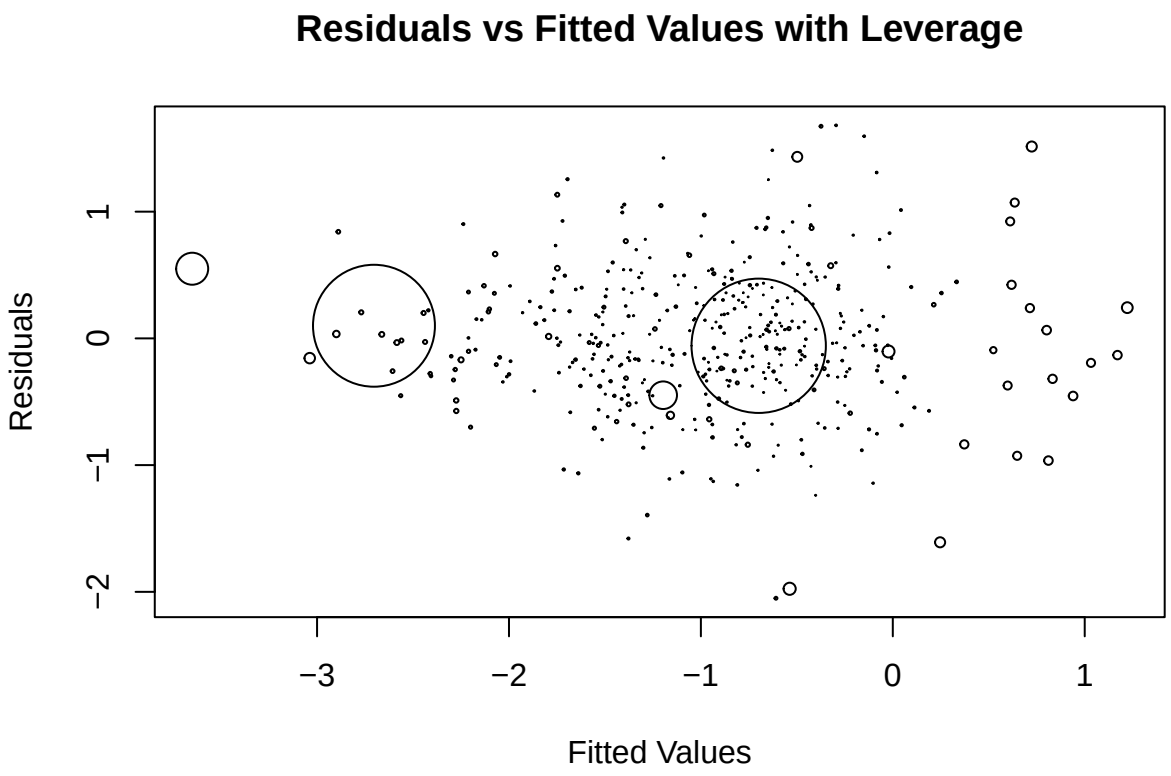



```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced

## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced
```



While the model selected based on the AIC criterion exhibits improved trends in terms of constant variance and linearity, it introduces an excessive number of interaction terms. This could potentially lead to overfitting and reduced interpretability. Given these considerations, lmod2 stands out as a more balanced and preferable candidate for our analysis. To ensure the robustness of our selected model, lmod2, it's essential to inspect potential leverage points that might unduly influence our regression results.

```
# Compute the leverage points
X = model.matrix(lmod2)
leverage = diag(X%*%solve(t(X)%*%X,t(X)))

# Plot residuals against fitted values, with point size indicating leverage
plot(lmod2$fit, lmod2$resid, cex=10*leverage, xlab="Fitted Values", ylab="Residuals", main="Residuals vs
```

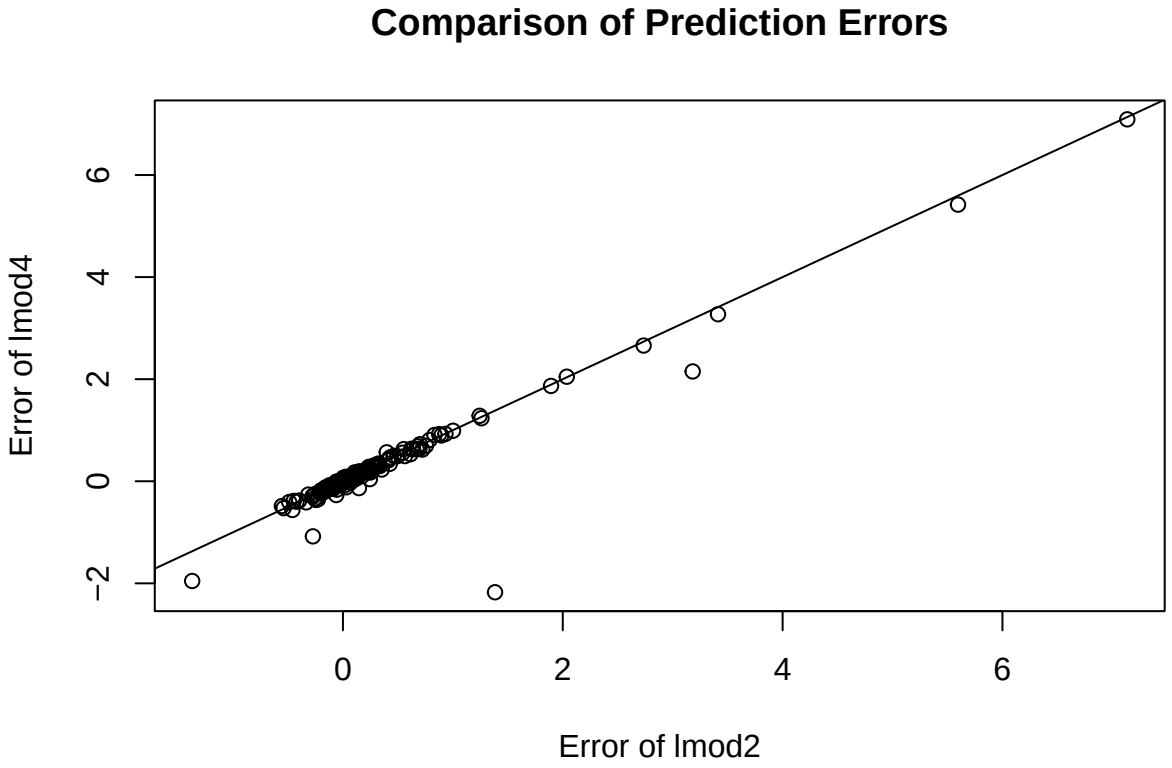


From the plot, we can identify two points with high leverage. However, their residuals are not exceptionally large, indicating that while these points might have a unique combination of predictor values, they are not significantly influencing the regression line. Thus, they are not considered problematic.

Having validated lmod2 against potential issues, it's now time to compare it with another model, lmod4, using a validation set.

First, we'll compute the predicted values for both models:

```
pred_lmod2 = exp(predict(lmod2, obsidian[validation,]))
pred_lmod4 = exp(predict(lmod4, obsidian[validation,]))

err_mod12 = obsidian[validation,]$mass - pred_lmod2
err_mod14 = obsidian[validation,]$mass - pred_lmod4

# Plotting the errors of the two models against each other
plot(err_mod12, err_mod14, xlab="Error of lmod2", ylab="Error of lmod4", main="Comparison of Prediction
abline(0, 1)
```



Next, we'll compute the mean squared error (MSE) and mean absolute error (MAE) for both models:

```
c(mean(err_mod12^2), mean(err_mod14^2))
```

```
## [1] 0.7034585 0.6863177
```

```
c(mean(abs(err_mod12)), mean(abs(err_mod14)))
```

```
## [1] 0.3481067 0.3543307
```

From the results, we observe that lmod4 has a lower MSE, while lmod2 boasts a lower MAE. This suggests that while lmod4 might be better at minimizing large errors, lmod2 is more consistent in its predictions.

To further assess the reliability of our models, we'll construct a 95% predictive interval for the mass in the validation set. If the coverage is significantly different from 95%, it indicates that model assumption violations might be compromising the reliability of our inferences.

```
# Construct 90% predictive intervals for both models
pred_int_model2 = exp(predict(lmod2, obsidian[validation,], interval='prediction', level=0.9))[,2:3]
pred_int_model4 = exp(predict(lmod4, obsidian[validation,], interval='prediction', level=0.9))[,2:3]

# Check the coverage of the predictive intervals
cover_model2 = (pred_int_model2[,1] <= obsidian[validation,]$mass) &
  (pred_int_model2[,2] >= obsidian[validation,]$mass)
cover_model4 = (pred_int_model4[,1] <= obsidian[validation,]$mass) &
  (pred_int_model4[,2] >= obsidian[validation,]$mass)

# Print the coverage percentages
mean(cover_model2)
```

```
## [1] 0.8916256
```

```r
mean(cover_model4)
```

```
## [1] 0.8817734
```

Both models appear to achieve satisfactory coverage levels. Overall, lmod2 demonstrates superior performance compared to lmod4.

Conclusion

Our final model is log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element_Zr). While this model isn't flawless, we have reasonable confidence in its validity given the moment conditions.