Final Data Analysis

2022-12-09

##1 Preliminary exploration and Data cleaning

First we read in the data and look at the summaries of the 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
```

We first notice that there seems to be some missing value. Let us look into that.

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

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

Data points in row 171 and 178 have missing values. For the time being, instead of doing an imputation, we omit these in order to increase the accuracy of the regression.

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

This is because we have enough number of the data sample, which is

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

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

Second thing we notice is abnormally high value of maximum of mass. We suspect that this data is corrupted.

```
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
```

Indeed, comparing with summary, we notice that the amount of four elements, Rb, Sr, Y, and Zr, does not seem to deviate too much from other data. By common sense, this indeed looks corrupted, so we delete this data point as well.

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

obsidian

Thirdly, if we look through obsidian data, which we omit due to space concern, some "type" labels are ambiguous.

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

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

Removing these 34 data points seems risky, because this can introduce bias. We want to merge by using the best guess, if possible. We can safely treat "Core fragment?," "Core fragment," "Cores and frags" and "Core/Fragment" as simply core. We can treat "Flake (listed as)" as simply Flake. We can treat "Distal end of prismatic blade?" and "Retouched blades" as simply Blade.

```
for (i in 1:dim(obsidian[1])){
  i = as.double(i)
  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"
  }
}
```

```
## Warning in 1:dim(obsidian[1]): numerical expression has 2 elements: only the
## first used
```

We also want to use consistent capital/small letter and singular/plural noun throughout the data in order to treat these as same when we do the regression.

```
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"
  }
}
```

Now, delete the rest data points with ambiguous type covariate.

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

By looking at the site covariate, we also notice the ambiguity given by "Ali Kosh/Chaga Sefid" in a data point 215. A data point 229 also has a site Hulailan Tepe Guran which is the only one in the site covariate. Since these are just two data points, it might be the best to remove these data points without being concerned too much about introducing bias.

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

We are now ready to do the regression.

##Model Selection

Let us first divide the data into a training set and validation set. We use training set to do the model selection and use validation set to avoid multiple testing issues and selective inference.

```
set.seed(1234)
n=dim(obsidian)[1]
ntrain = 410; nval = 203
```

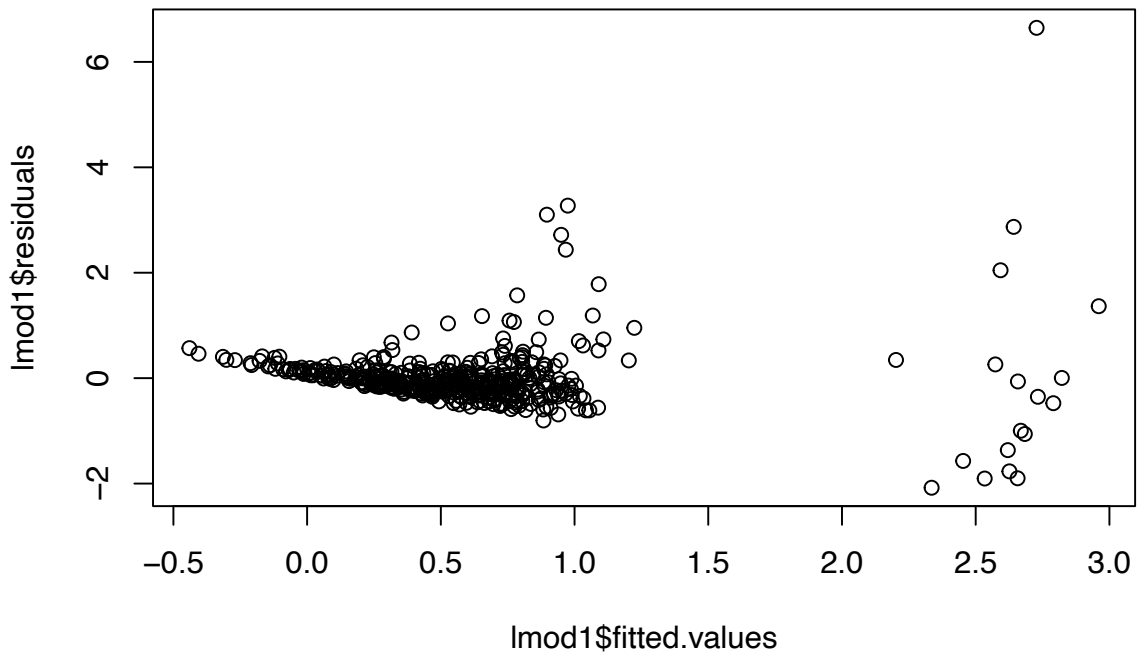
```
validation = sample(n, nval, replace=FALSE)
train = setdiff(1:n, validation)
```

First, we decide not to include ID as a covariate in our regression. Other than the obvious reason that each data point has a different ID, we do not see any visible pattern. For example, if we restrict ID to the first three digits, they will be exactly collinear with the site covariate. If we were to expand it to more than first three digits, again model gets too large, hence increasing variance too much.

```
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)
```

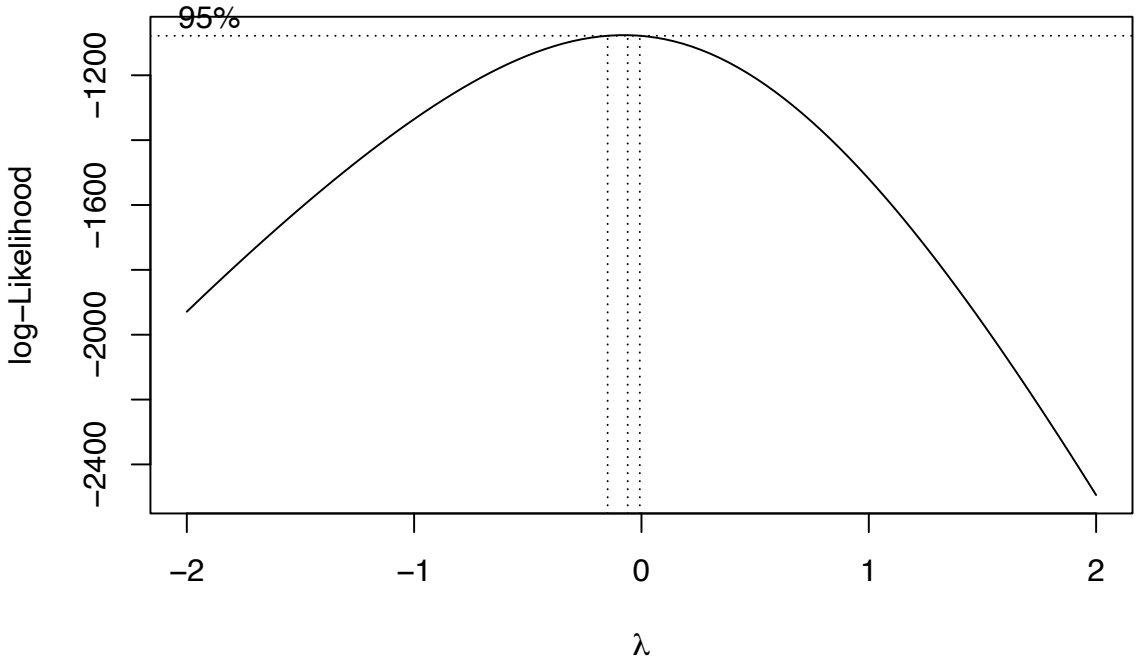
```
##
## Call:
## lm(formula = mass ~ type + site + element_Rb + element_Sr + element_Y +
##     element_Zr, data = obsidian[train, ])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0792 -0.2308 -0.0758  0.1152  6.6463
##
## Coefficients:
##                   Estimate Std. Error t 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
```

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



We observe nonlinearity and nonconstant variance issue at the same time, so instead of using weighted least square, we may consider transformation of data. Let us compare log transformation and boxcox transformation.

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




```
bc
```

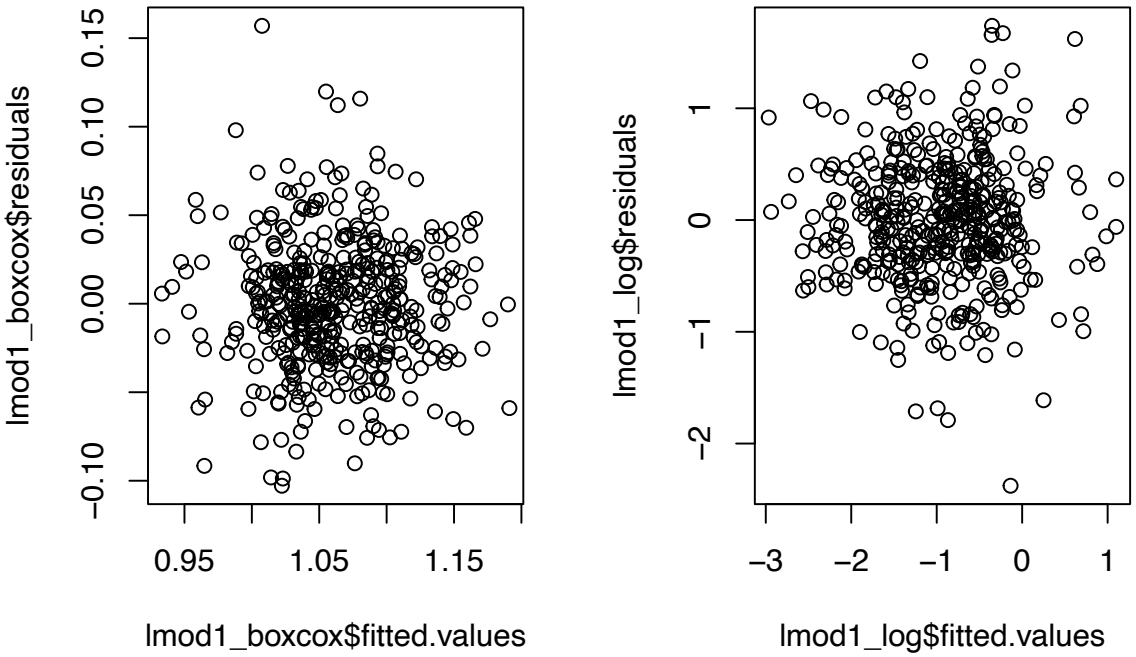
```
## $x
##   [1] -2.00000000 -1.95959596 -1.91919192 -1.87878788 -1.83838384 -1.79797980
##   [7] -1.75757576 -1.71717172 -1.67676768 -1.63636364 -1.59595960 -1.55555556
##  [13] -1.51515152 -1.47474747 -1.43434343 -1.39393939 -1.35353535 -1.31313131
##  [19] -1.27272727 -1.23232323 -1.19191919 -1.15151515 -1.11111111 -1.07070707
```

```
##  [25] -1.03030303 -0.98989899 -0.94949495 -0.90909091 -0.86868687 -0.82828283
##  [31] -0.78787879 -0.74747475 -0.70707071 -0.66666667 -0.62626263 -0.58585859
##  [37] -0.54545455 -0.50505051 -0.46464646 -0.42424242 -0.38383838 -0.34343434
##  [43] -0.30303030 -0.26262626 -0.22222222 -0.18181818 -0.14141414 -0.10101010
##  [49] -0.06060606 -0.02020202  0.02020202  0.06060606  0.10101010  0.14141414
##  [55]  0.18181818  0.22222222  0.26262626  0.30303030  0.34343434  0.38383838
##  [61]  0.42424242  0.46464646  0.50505051  0.54545455  0.58585859  0.62626263
##  [67]  0.66666667  0.70707071  0.74747475  0.78787879  0.82828283  0.86868687
##  [73]  0.90909091  0.94949495  0.98989899  1.03030303  1.07070707  1.11111111
##  [79]  1.15151515  1.19191919  1.23232323  1.27272727  1.31313131  1.35353535
##  [85]  1.39393939  1.43434343  1.47474747  1.51515152  1.55555556  1.59595960
##  [91]  1.63636364  1.67676768  1.71717172  1.75757576  1.79797980  1.83838384
##  [97]  1.87878788  1.91919192  1.95959596  2.00000000
##
## $y
##   [1] -1928.988 -1901.890 -1874.991 -1848.299 -1821.821 -1795.561 -1769.527
##   [8] -1743.725 -1718.164 -1692.850 -1667.794 -1643.002 -1618.486 -1594.255
##  [15] -1570.319 -1546.690 -1523.381 -1500.404 -1477.773 -1455.503 -1433.609
##  [22] -1412.109 -1391.020 -1370.361 -1350.152 -1330.416 -1311.175 -1292.453
##  [29] -1274.276 -1256.672 -1239.669 -1223.298 -1207.591 -1192.580 -1178.301
##  [36] -1164.789 -1152.081 -1140.216 -1129.233 -1119.170 -1110.067 -1101.963
##  [43] -1094.897 -1088.907 -1084.029 -1080.297 -1077.745 -1076.400 -1076.291
##  [50] -1077.439 -1079.865 -1083.585 -1088.608 -1094.945 -1102.597 -1111.564
##  [57] -1121.843 -1133.426 -1146.302 -1160.456 -1175.873 -1192.535 -1210.420
##  [64] -1229.506 -1249.769 -1271.186 -1293.729 -1317.374 -1342.092 -1367.857
##  [71] -1394.640 -1422.415 -1451.154 -1480.827 -1511.408 -1542.869 -1575.182
##  [78] -1608.318 -1642.252 -1676.954 -1712.400 -1748.561 -1785.412 -1822.926
##  [85] -1861.079 -1899.845 -1939.200 -1979.121 -2019.583 -2060.565 -2102.044
##  [92] -2143.999 -2186.410 -2229.256 -2272.519 -2316.180 -2360.222 -2404.627
##  [99] -2449.380 -2494.462
```

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

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

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

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)
```



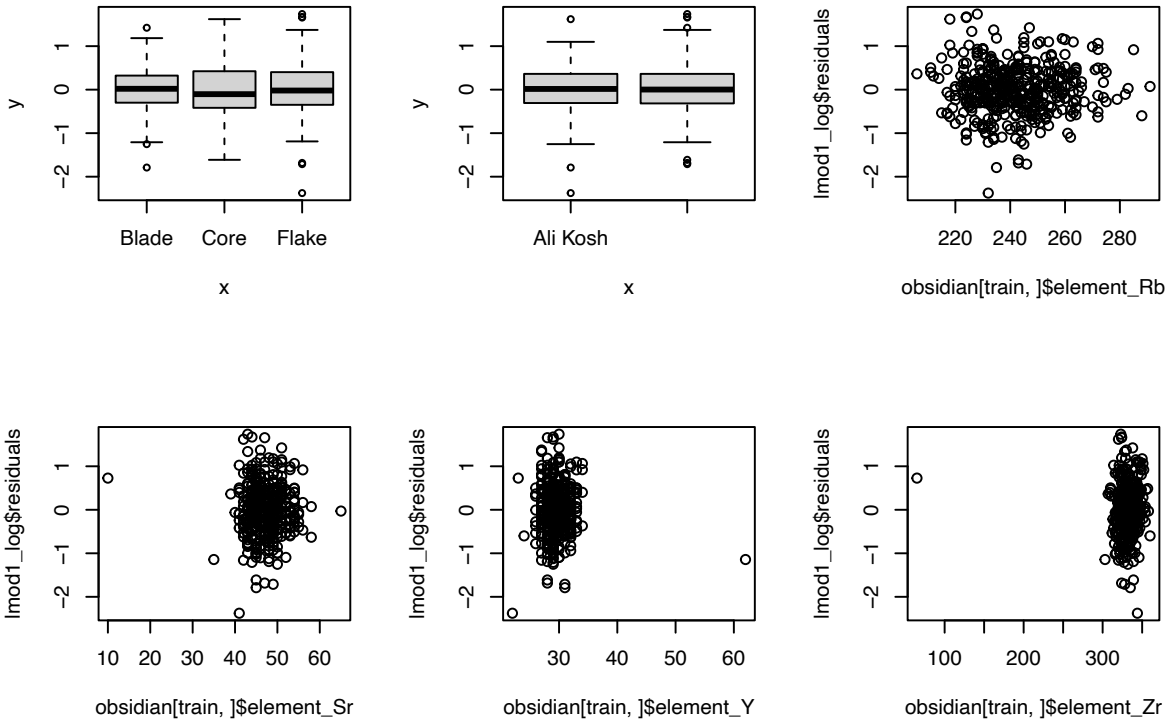
Both look great in terms of our linearity/constant variance assumptions, so we may choose log transformation, which is more natural.

```
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
```

Remember that intercept term implicitly considers “Blade” type as a reference level, so even though t value of “Flake” type covariate is high, we do not want to delete it. This does seem like a valid model. Let us check with the validation data set.

```
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)
```



There are some outliers, which we should keep an eye on, but overall linearity and constant variance assumption seem to follow for this model.

```
BIC(lmod1_log)
```

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

So then it is natural to ask if there is an interaction term. We have $\binom{6}{2} = 15$ many two-way interactions to try. We use 0.05/15=0.003 threshold.

```
BIC(lm(log(mass) ~ (type * site + element_Rb + element_Sr + element_Y + element_Zr), data = obsidian[tra
```

```
## [1] 803.2455
```

```
BIC(lm(log(mass) ~ (type * element_Rb + site + element_Sr + element_Y + element_Zr), data = obsidian[tra
```

```
## [1] 808.7675
```

```
BIC(lm(log(mass) ~ (type * element_Sr + site + element_Rb + element_Y + element_Zr), data = obsidian[tra
```

```
## [1] 811.1033
```

```
BIC(lm(log(mass) ~ (type * element_Y + site + element_Rb + element_Sr + element_Zr), data = obsidian[tra
```

```
## [1] 803.1618
```

```
BIC(lm(log(mass) ~ (type * element_Zr + site + element_Rb + element_Sr + element_Y), data = obsidian[tra
```

```
## [1] 807.427
```

```
BIC(lm(log(mass) ~ (type + site * element_Rb + element_Sr + element_Y + element_Zr), data = obsidian[tra
```

```
## [1] 805.2066
```

```
BIC(lm(log(mass) ~ (type + site * element_Sr + element_Rb + element_Y + element_Zr), data = obsidian[tra
```

```
## [1] 804.9216
```

```
BIC(lm(log(mass) ~ (type + site * element_Y + element_Rb + element_Sr + element_Zr), data = obsidian[tra
```

```
## [1] 798.3041
```

```
BIC(lm(log(mass) ~ (type + site * element_Zr + element_Rb + element_Sr + element_Y), data = obsidian[tra
```

```
## [1] 804.2896
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb * element_Sr + element_Y + element_Zr), data = obsidian[tra
```

```
## [1] 799.8755
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb * element_Y + element_Sr + element_Zr), data = obsidian[tra
```

```
## [1] 792.7635
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb * element_Zr + element_Sr + element_Y), data = obsidian[tra
```

```
## [1] 783.486
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb + element_Sr * element_Y + element_Zr), data = obsidian[tra
```

```
## [1] 786.9142
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb + element_Sr * element_Zr + element_Y), data = obsidian[tra
```

```
## [1] 790.2093
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element_Zr), data = obsidian[tra
```

```
## [1] 770.7243
```

We add the interaction term element_Y * element_Zr.

```
BIC(lm(log(mass) ~ (type * site + element_Rb + element_Sr + element_Y + element_Y * element_Zr), data =
```

```
## [1] 776.3758
```

```
BIC(lm(log(mass) ~ (type * element_Rb + site + element_Sr + element_Y + element_Y * element_Zr), data =
```

```
## [1] 781.9062
```

```
BIC(lm(log(mass) ~ (type * element_Sr + site + element_Rb + element_Y + element_Y * element_Zr), data =
```

```
## [1] 782.3556
```

```
BIC(lm(log(mass) ~ (type * element_Y + site + element_Rb + element_Sr + element_Y * element_Zr), data =
```

```
## [1] 779.9719
```

```
BIC(lm(log(mass) ~ (type * element_Zr + site + element_Rb + element_Sr + element_Y * element_Zr), data =
```

```
## [1] 779.1342
```

```
BIC(lm(log(mass) ~ (type + site * element_Rb + element_Sr + element_Y + element_Y * element_Zr), data =
```

```
## [1] 776.7671
```

```
BIC(lm(log(mass) ~ (type + site * element_Sr + element_Rb + element_Y + element_Y * element_Zr), data =
```

```
## [1] 776.3978
```

```
BIC(lm(log(mass) ~ (type + site * element_Y + element_Rb + element_Sr + element_Y * element_Zr), data =
```

```
## [1] 776.7883
```

```
BIC(lm(log(mass) ~ (type + site * element_Zr + element_Rb + element_Sr + element_Y * element_Zr), data
```

```
## [1] 776.7353
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb * element_Sr + element_Y + element_Y * element_Zr), data =
```

```
## [1] 775.3176
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb * element_Y + element_Sr + element_Y * element_Zr), data =
```

```
## [1] 776.541
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb * element_Zr + element_Sr + element_Y * element_Zr), data
```

```
## [1] 772.6743
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb + element_Sr * element_Y + element_Y * element_Zr), data =
```

```
## [1] 773.2828
```

```
BIC(lm(log(mass) ~ (type + site + element_Rb + element_Sr * element_Zr + element_Y * element_Zr), data
```

```
## [1] 776.7995
```

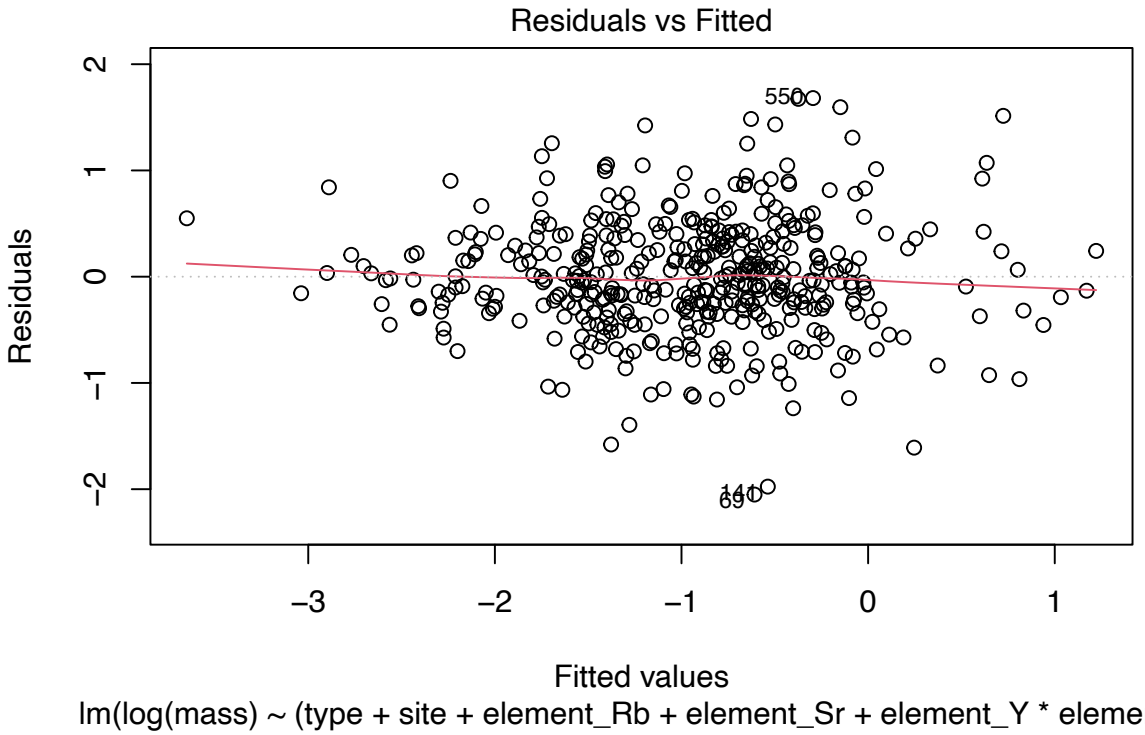
We do not further add an interaction term since 770 is less than all of the above.

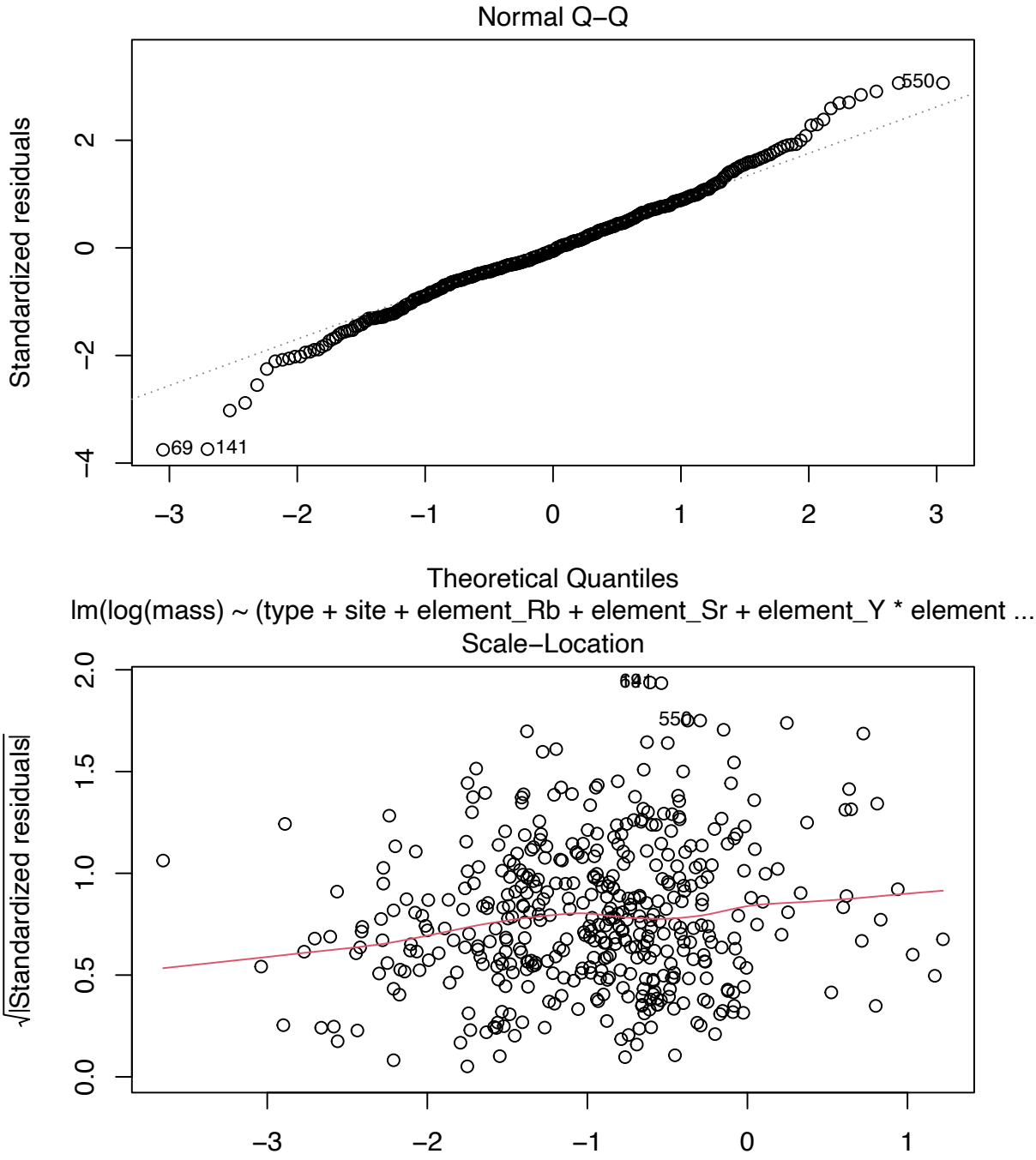
```
lmod2 <- lm(log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element_Zr), data = obsidi
summary(lmod2)
```

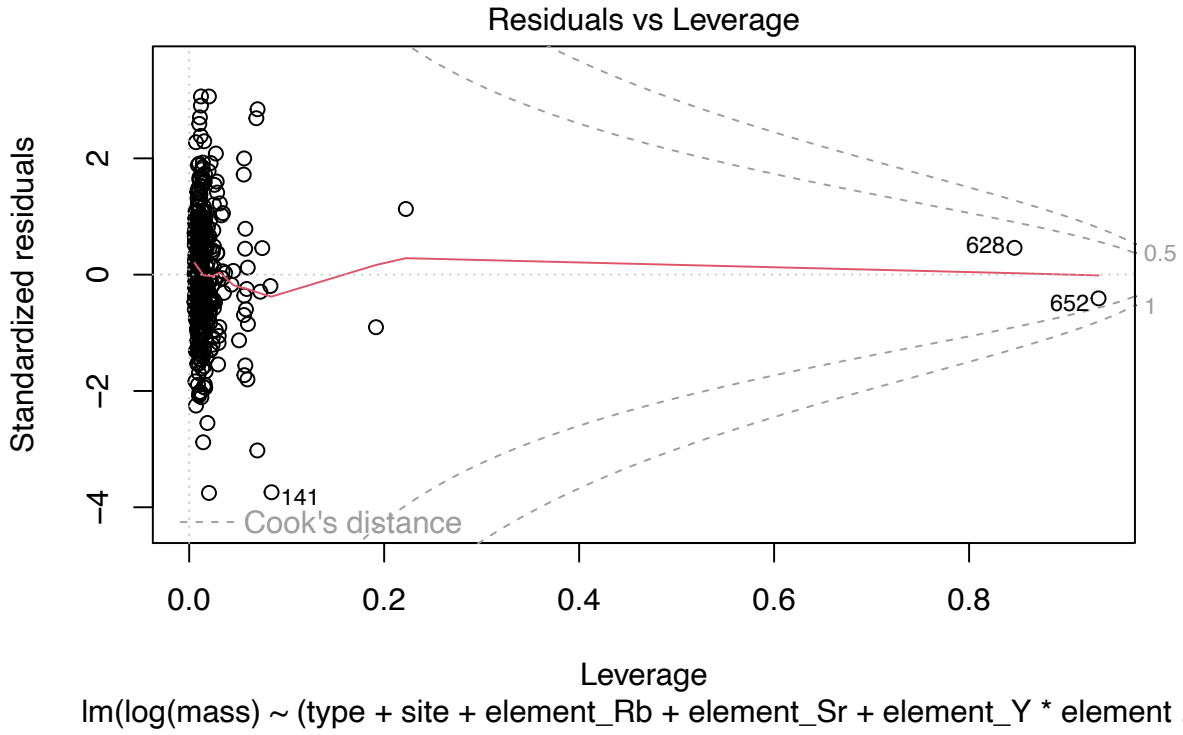
```
##
## Call:
## lm(formula = log(mass) ~ (type + site + element_Rb + element_Sr +
##     element_Y * element_Zr), data = obsidian[train, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.05050 -0.30275 -0.03086  0.33680  1.68102
##
## Coefficients:
##                          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 ...
Scale–Location
√|Standardized residuals|
Fitted values
lm(log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element ...



We do see a bit of nonconstant variance trend toward the both ends and nonlinearity trend, but they are not severe. We compare this model with AIC criterion, using step function as follows.

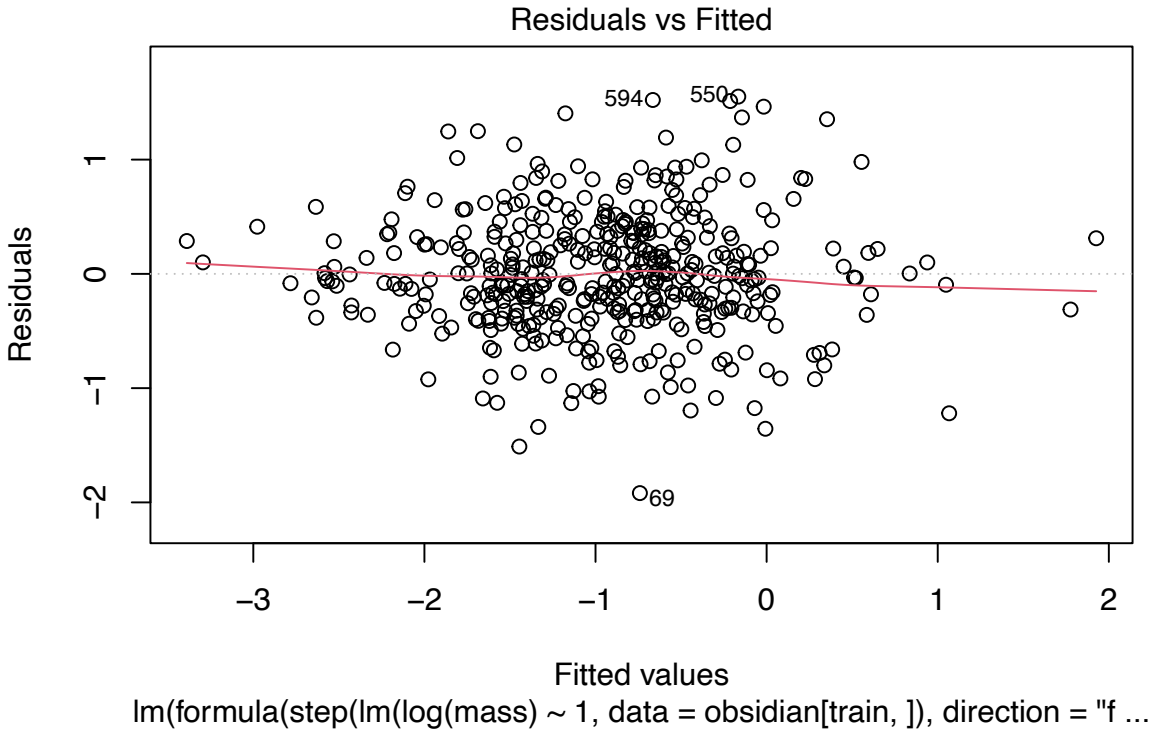
```
lmod3 <- lm(log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element_Zr)**2, data = obs
lmod4 <- lm(formula(step(lm(log(mass)~1, data = obsidian[train, ]), direction='forward', scope=formula(
summary(lmod4)
```

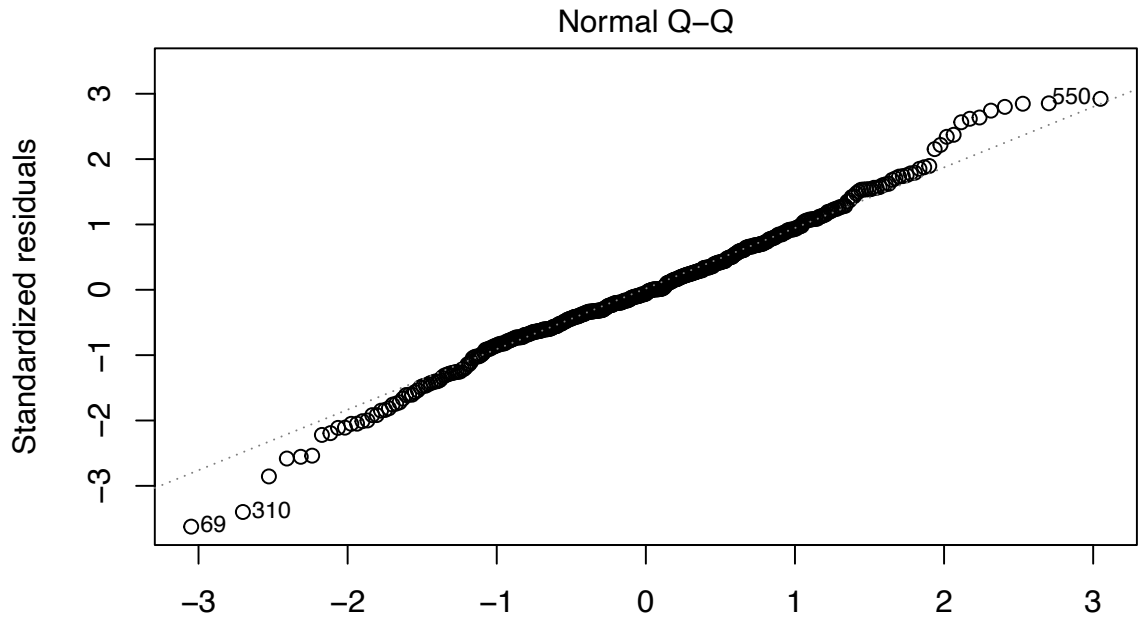
```
##
## Call:
## lm(formula = formula(step(lm(log(mass) ~ 1, data = obsidian[train,
##     ]), direction = "forward", scope = formula(lmod3), trace = 0)),
##     data = obsidian[train, ])
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.91858 -0.32215 -0.03133  0.32270  1.55136
##
## 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
```

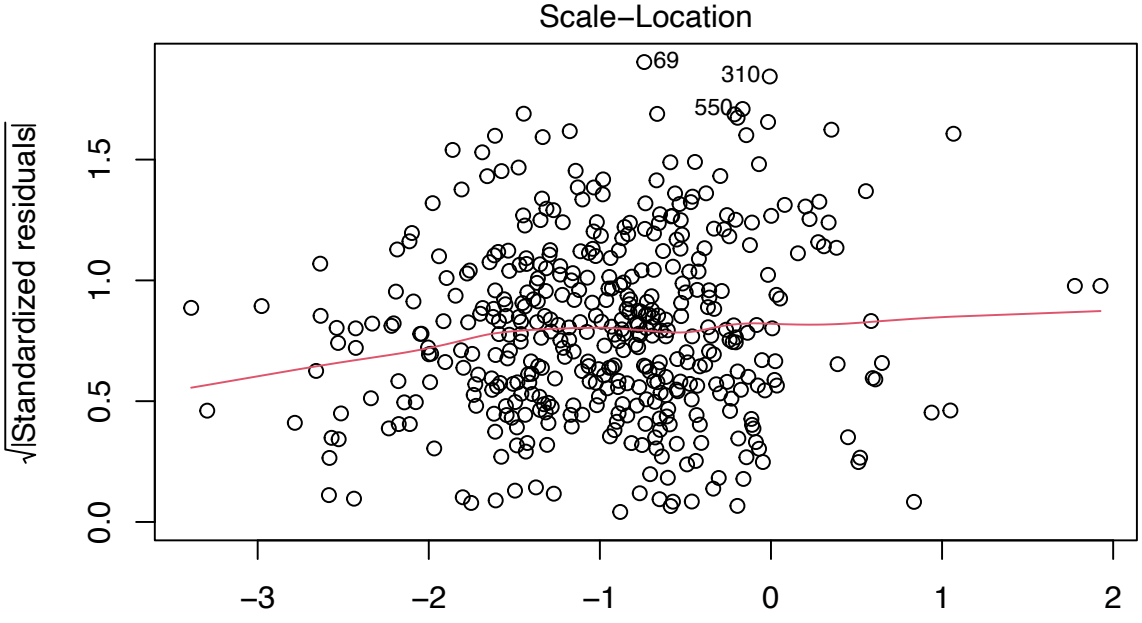
```
## element_Zr:element_Y        0.0063046  0.0028439   2.217 0.027170 *
## typeCore:element_Y          0.3675345  0.1733637   2.120 0.034594 *
## typeFlake:element_Y         0.1029781  0.0524898   1.962 0.050443 .
## element_Rb:typeCore        -0.0167357  0.0237771  -0.704 0.481915
## element_Rb:typeFlake       -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 expected, this model is more lenient than BIC so the size is larger than lmod2.

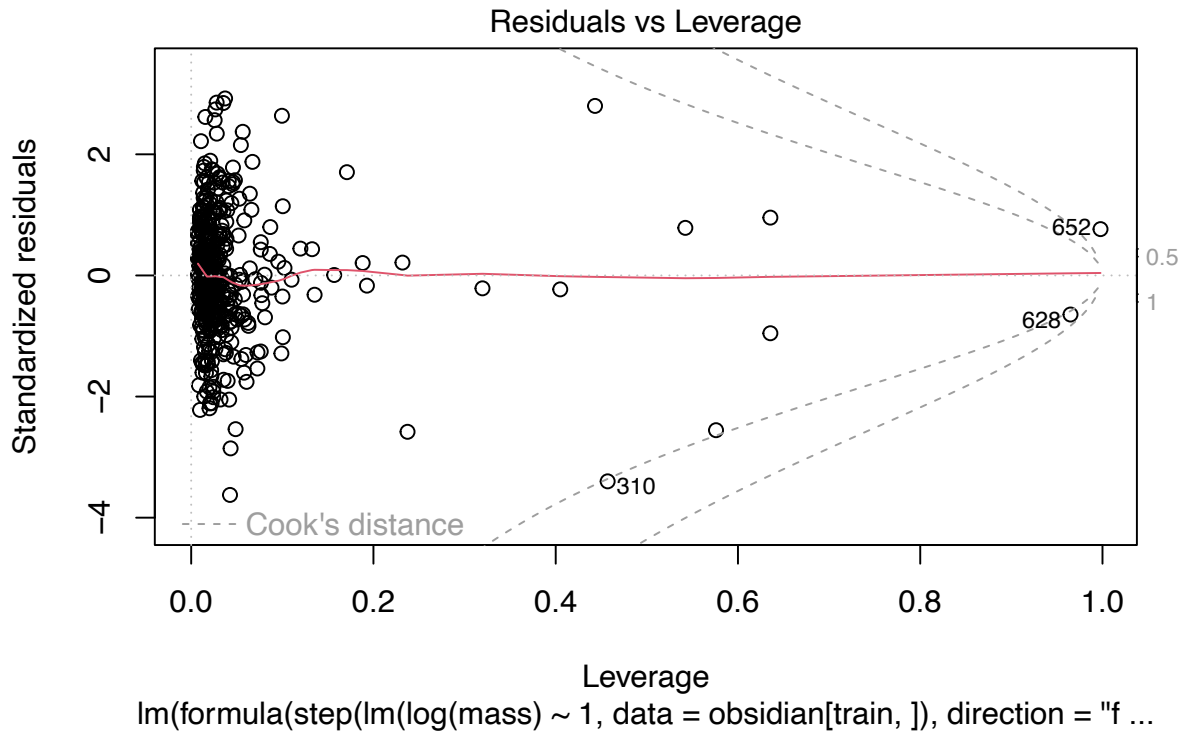
```
plot(lmod4)
```







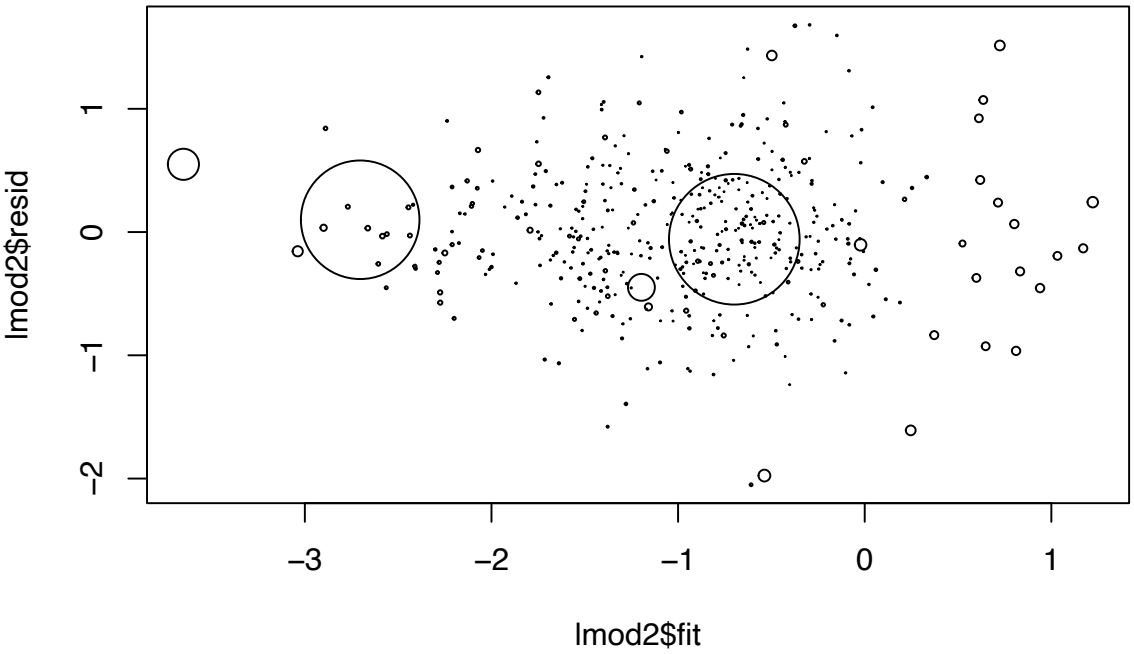

```
## Warning in sqrt(crit * p * (1 - hh)/hh): NaNs produced

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



Clearly, we have a better constant variance/linearity trend, but we have added too many interaction terms. lmod2 is a great candidate for our model.

Let us check leverage points.

```
X = model.matrix(lmod2)
leverage = diag(X%*%solve(t(X)%*%X,t(X)))
plot(lmod2$fit,lmod2$resid,cex=10*leverage)
```

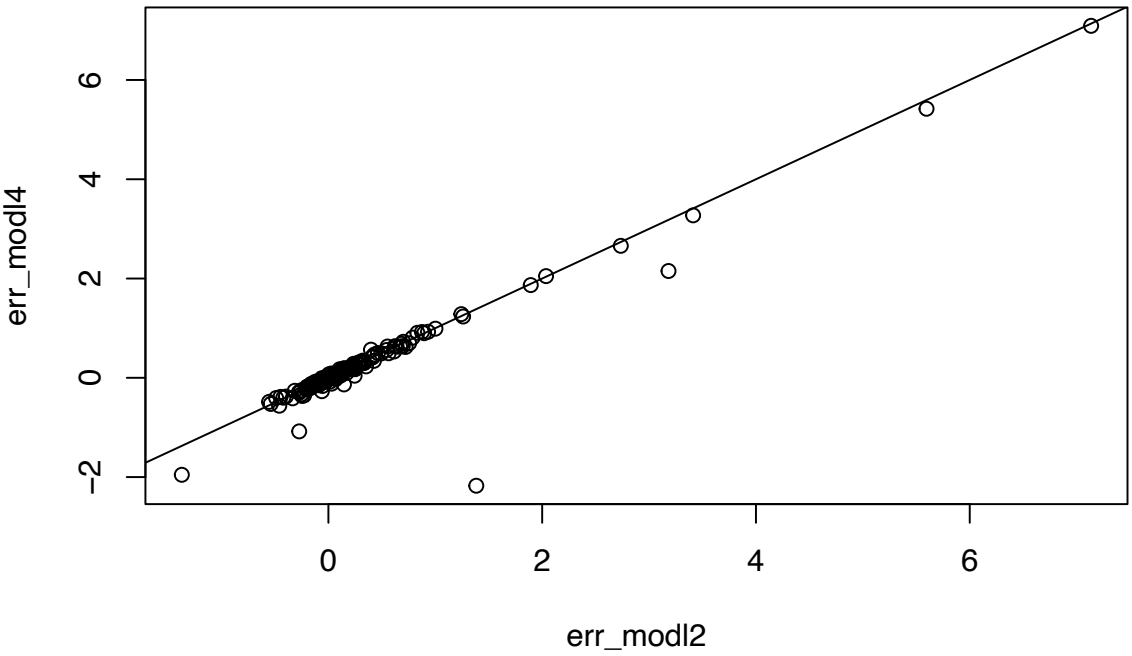


We do observe two huge leverage points but they are not too worrisome since they are not outliers (do not

have a huge residual). Now, lmod2 seems to be a great cadidate model for our data. Let's compare lmod2 and lmod4 with a validation set.

First we can compare in terms of predictive error:

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

err_modl2 = obsidian[validation,]$mass - pred_lmod2
err_modl4 = obsidian[validation,]$mass - pred_lmod4

plot(err_modl2, err_modl4)
abline(0, 1)
```




```
c(mean(err_modl2^2), mean(err_modl4^2))
```

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

```
c(mean(abs(err_modl2)), mean(abs(err_modl4)))
```

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

We see that for mean square error, modl4 has a lower error, but for mean absolute error, modl2 has a lower error.

We can also verify validity of inference for our models. For each model, we will construct a 95% predictive interval for mass in the validation set. We will then check for 95% coverage—if it fails empirically, then this means that model assumption violations are causing our inference to be unreliable.

```
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]
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)
mean(cover_model2)
```

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

```
mean(cover_model4)
```

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

Both model seem to be achieving an appropriate coverage level. In particular, model2 seem to perform better than model4.

Our final model is thus log(mass) ~ (type + site + element_Rb + element_Sr + element_Y * element_Zr) While this is not perfect, we are reasonably confident that the model assumptions hold, at least for the moment conditions.