Moloco Data Assignment

Data loading

```
##
## Attaching package: 'plotly'

## The following object is masked from 'package:ggplot2':
##
##     last_plot

## The following object is masked from 'package:stats':
##
##     filter

## The following object is masked from 'package:graphics':
##
##     layout

## Loading required package: lattice

## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric

##
## Attaching package: 'lmtest'

## The following object is masked from 'package:RCurl':
##
##     reset

## Parsed with column specification:
## cols(
##   ts = col_datetime(format = ""),
##   user_id = col_character(),
##   country_id = col_character(),
##   site_id = col_character()
## )
```

For ease of use, I downloaded and converted the data to CSV as it preserves the shape and properties of the data. I chose to use the `data.table` library to do all my operations as it provides a simple interface to access data and for large datasets is massively parallel

Part 1

```r
cbdv <- data[country_id == 'BDV']
ans = unique(
  (
    as.data.table(cbdv)[, count := uniqueN(user_id), by = site_id]
    )[, c('site_id', 'count'), with = FALSE]
  )[order(-count)]
print(ans)
```

```
##    site_id count
## 1:   5NPAU   544
## 2:   N0OTG    90
## 3:   3P0LC     2
```

This code actually forms the base for most of the assignment. First we filter by country - we could do this inline, but it is easier to read this way - then we simply group by `site_id` and count the number of unique `user_id`s we have. We then clean up the result by dropping everything except our `site_id` and `count`. After removing duplicates, due to how the grouping in `data.table` works, and ordering by count, we print the result.

Part 2

```
tdat <- data[between(ts, as.POSIXct("2019-02-03 00:00:00", tz="UTC"), as.POSIXct("2019-02-04 23:59:59",
ans2 = ((as.data.table(tdat)[, .(`number of visits` = .N), by = list(user_id, site_id)])[`number of vis
print(ans2)
```

```
##    user_id site_id number of visits
## 1:  LC3A59   N0OTG               26
## 2:  LC06C3   N0OTG               25
## 3:  LC3C9D   N0OTG               17
## 4:  LC3C7E   3P0LC               15
```

We again follow the same pattern, beginning by filtering. Here it is noticeable that the initial filter on a seperate line is useful because it is quite cumbersome with its `between` statement. After this, it is a simple matter of grouping on multiple columns: both `user_id` and `site_id`, and counting how many of those rows there are. `data.table` gives us a simple interface to do this. Finally, we filter by `number of visits`, order by visits, and print.

Part 3

```
udat <- data[, .SD[which.max(ts)], by=user_id]
ans3 <- unique(
  (
    as.data.table(udat)[ , count := uniqueN(user_id), by = site_id]
    )[, c('site_id', 'count'), with = FALSE]
  )[order(-count)]
print(ans3)
```

```
##    site_id count
## 1:   5NPAU   992
## 2:   N0OTG   561
## 3:   QG03G   289
## 4:   GV0FK    42
## 5:   3P0LC    28
## 6:   RT9Z6     2
## 7:   JSUUP     1
## 8:   EUZ/Q     1
```

To find the last visited site, we can use a little trick and simply get the maximum timestamp for each user, which is what the `.SD` in `data.table` lets us do. By creating that first filter, we are able to reuse the code from Part 1 to do our group, cleanup, and ordering to then get our result.

Part 4

```r
udatFirst <- data[, .SD[which.min(ts)], by=user_id][, c('site_id', 'user_id'), with = FALSE]
udatLast <- data[, .SD[which.max(ts)], by=user_id][, c('site_id', 'user_id'), with = FALSE]
count <- nrow(merge(udatFirst, udatLast))
print(count)
```

```
## [1] 1670
```

Here we use a little trick of `data.table`. First we use the code from part 3 to get both the minimum and maximum timestamp for each user. This is the same as their first and last visit. What we can do next is merge the resultant tables. The key here is that in its default behavior, `data.table` will keep rows that exist in both tables, essentially the users who's first and last visit was the same site. This is much more efficient than looping over both tables because table merges can be done in parallel, rather than a linear scan. Finally, we just count the number of rows in our merge, and print the result.

Part 5

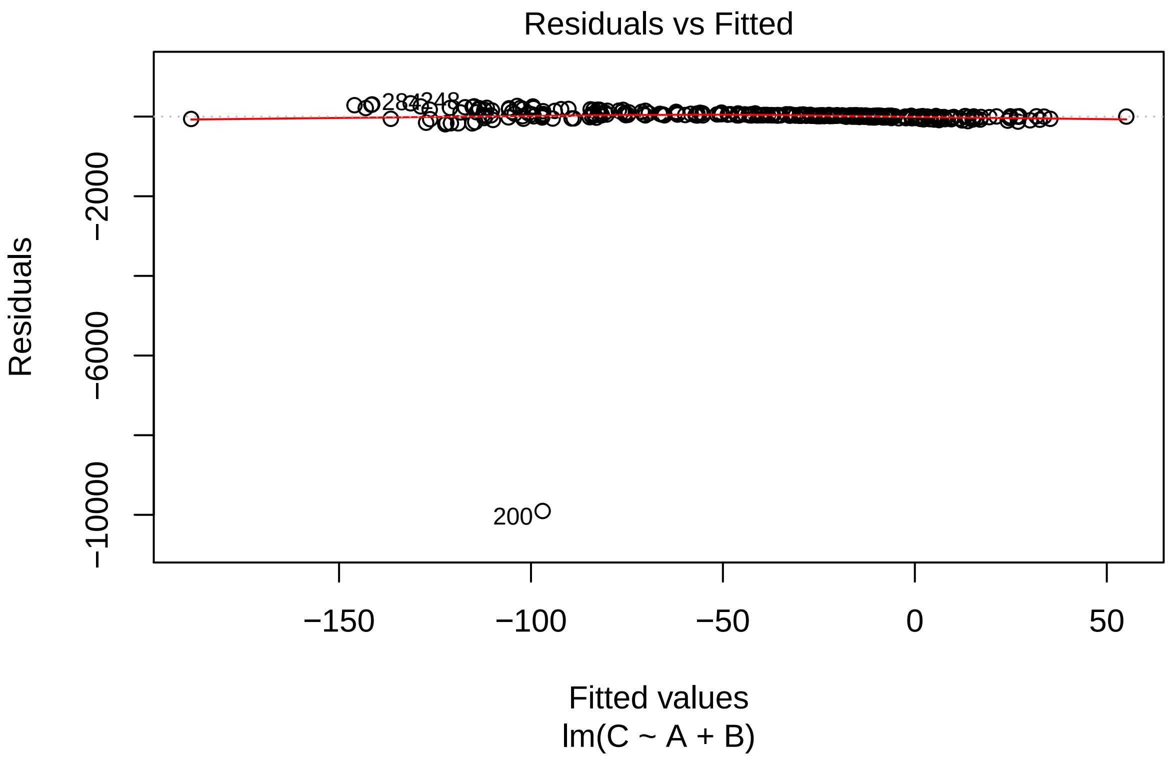
```r
data = read_csv("regression.csv")
```

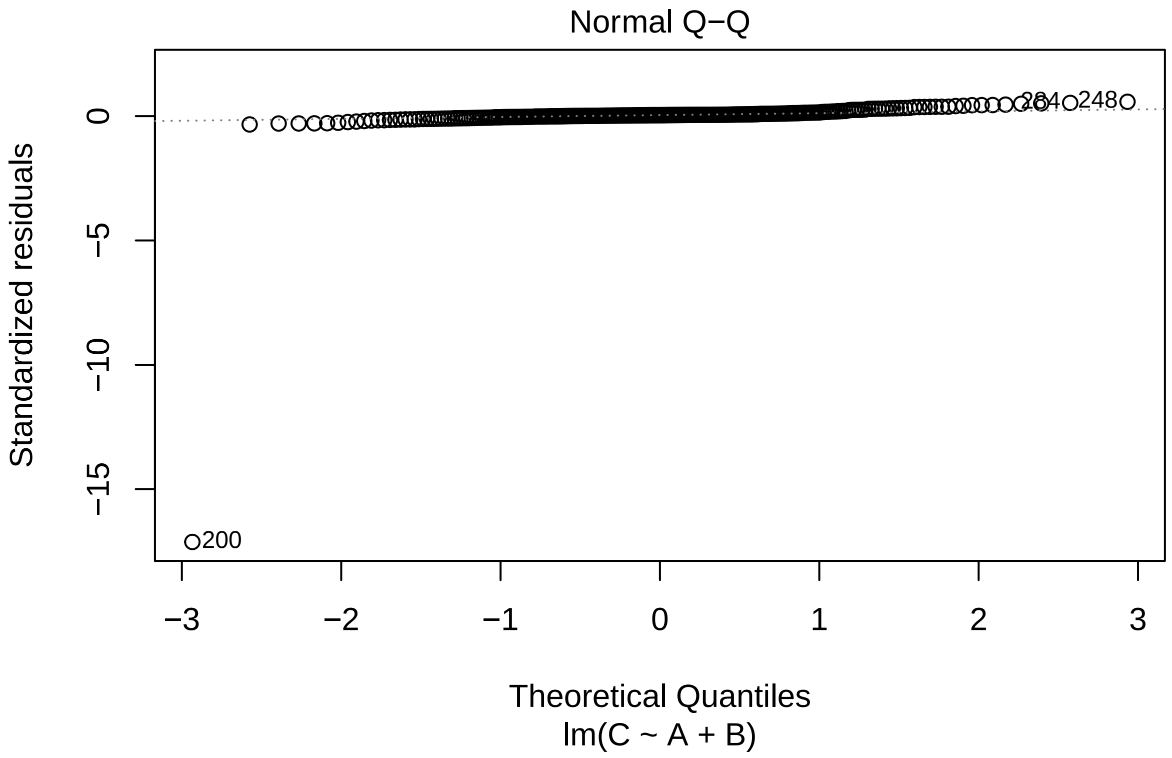
```
## Parsed with column specification:
## cols(
##   `4.90E-01` = col_double(),
##   `-1.80E-01` = col_double(),
##   `1.15E+01` = col_double()
## )
```

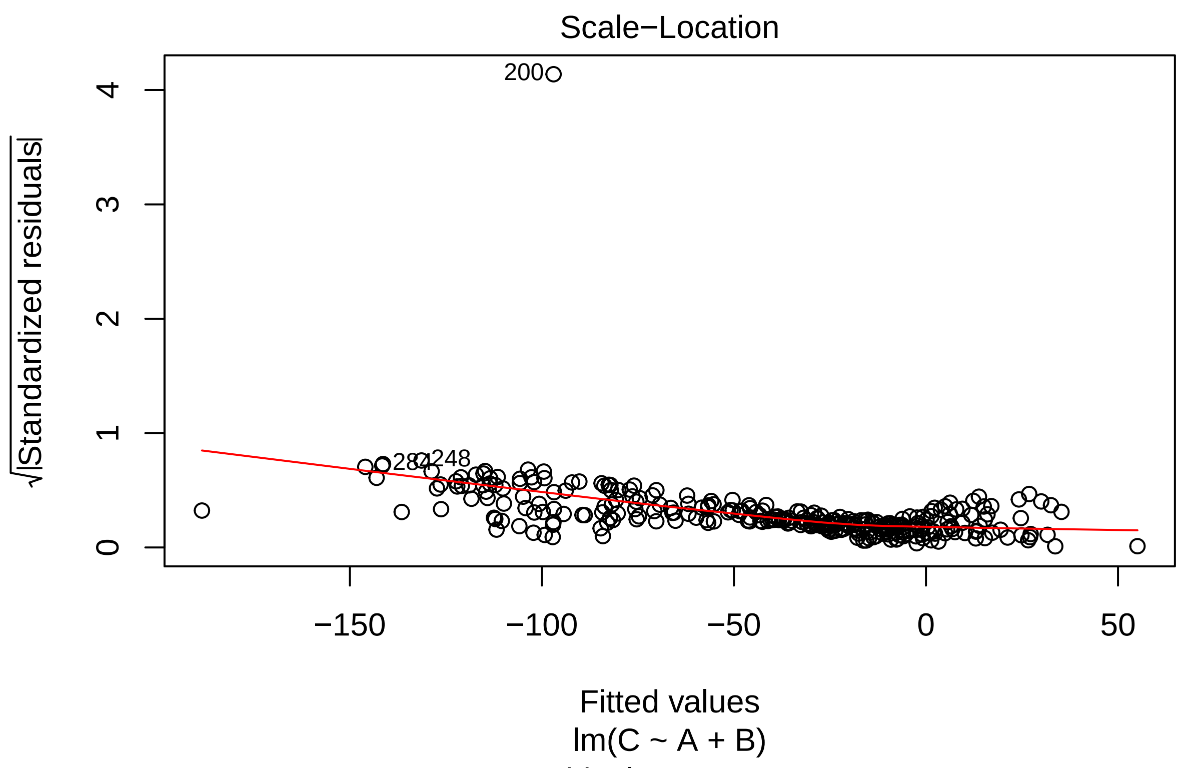
```r
names(data) = c("A", "B", "C")
reg_model = lm(C~A + B, data = data)
summary(reg_model)
```

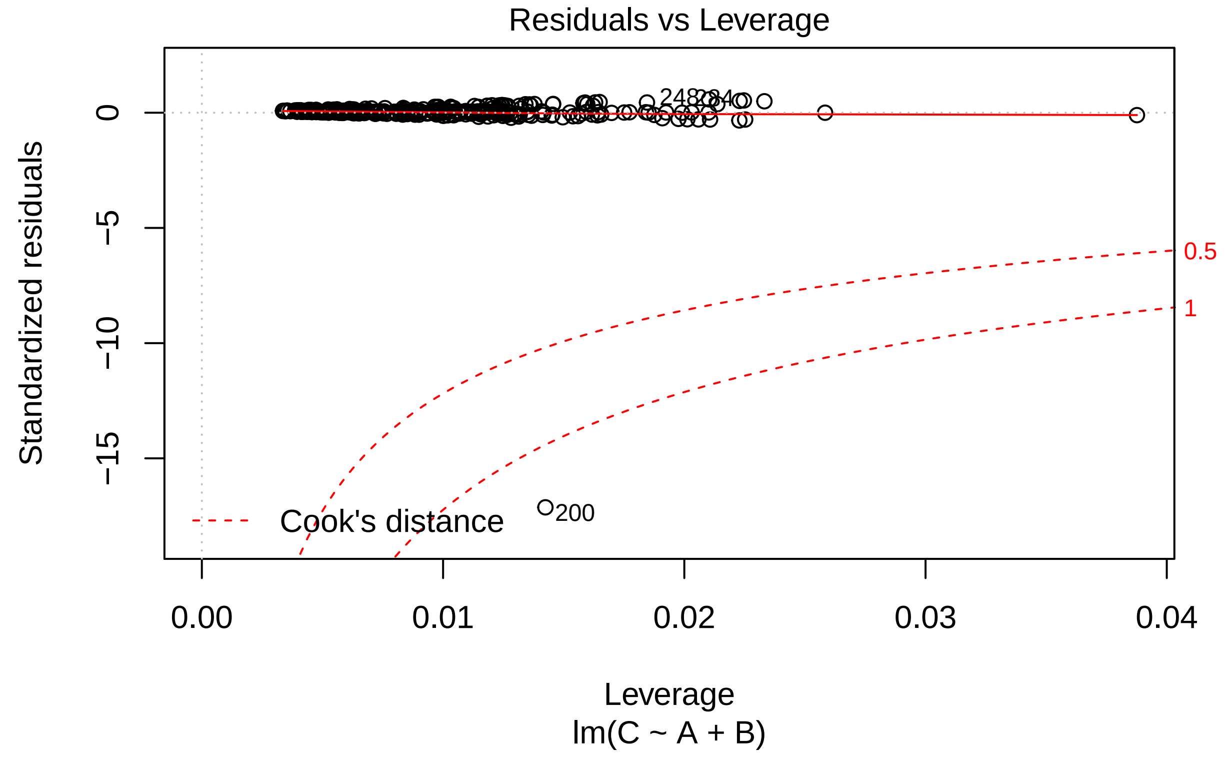
```
##
## Call:
## lm(formula = C ~ A + B, data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9903.0    -5.2    23.0    54.1   333.3
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -19.496     44.632  -0.437    0.663
## A             -8.057      6.029  -1.336    0.182
## B             -1.746     11.459  -0.152    0.879
##
## Residual standard error: 582.4 on 296 degrees of freedom
## Multiple R-squared:  0.006037,   Adjusted R-squared:  -0.0006787
## F-statistic: 0.8989 on 2 and 296 DF,  p-value: 0.4081
```

```
plot(reg_model)
```





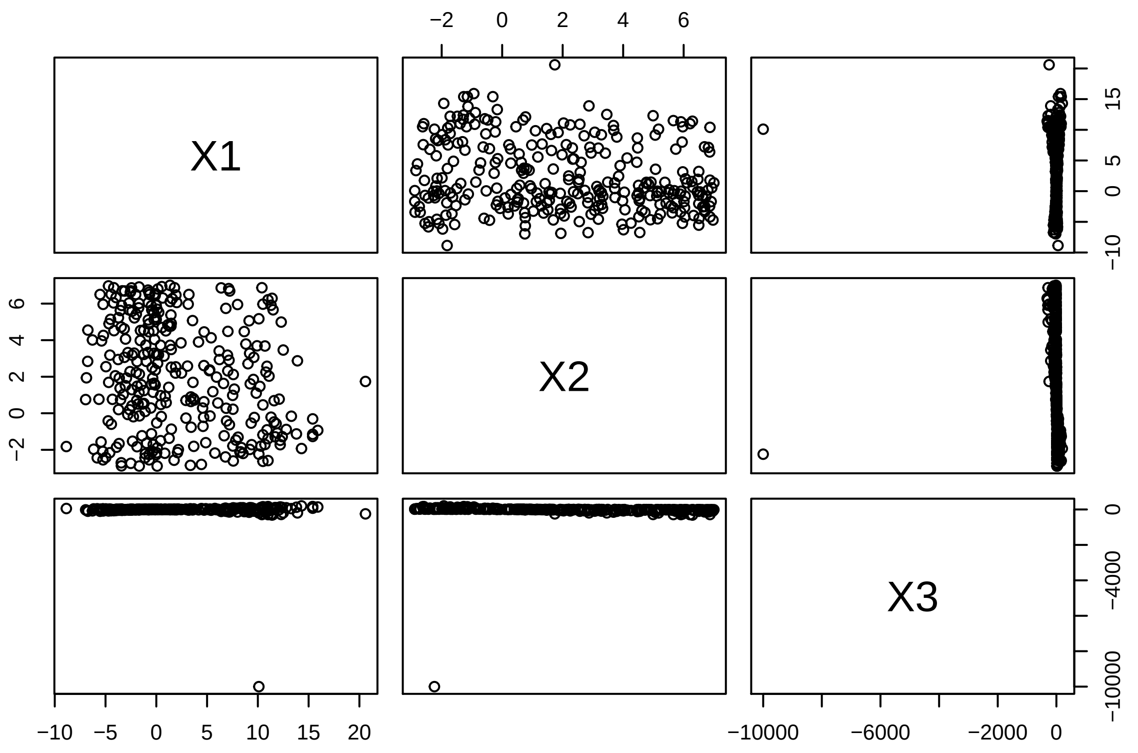




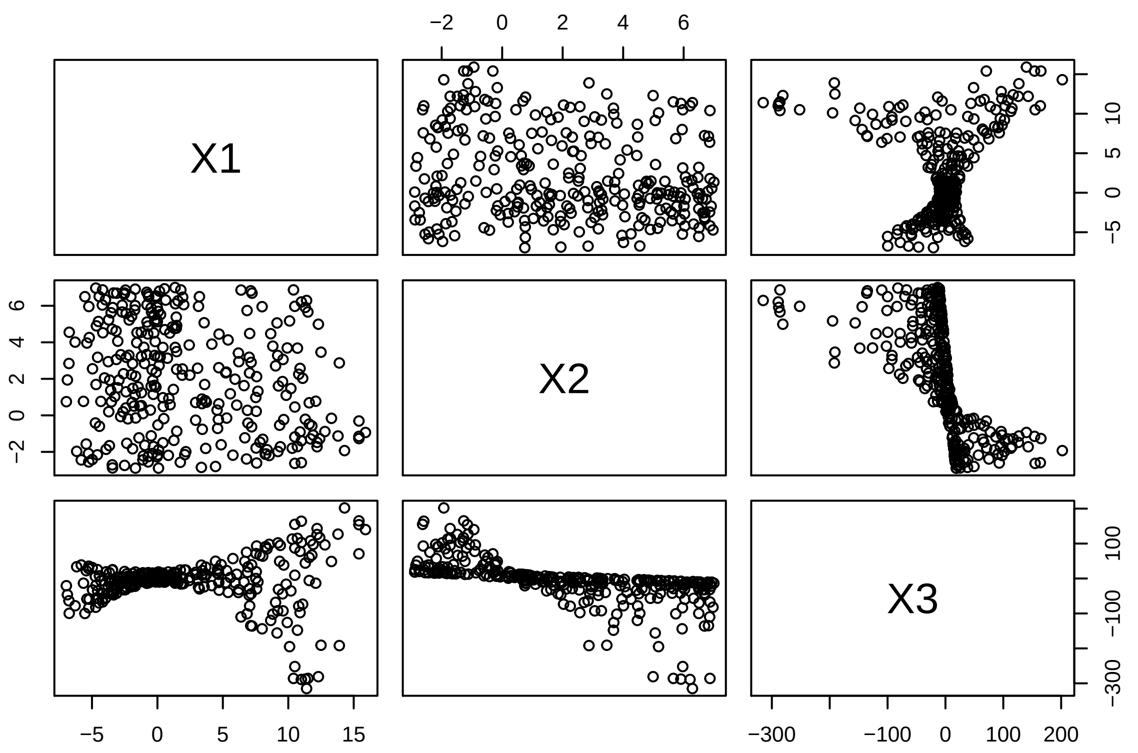
```
data <- read_csv("regression.csv", col_names = FALSE)
```

```
## Parsed with column specification:
## cols(
##   X1 = col_double(),
##   X2 = col_double(),
##   X3 = col_double()
## )
```

```
plot(data)
```



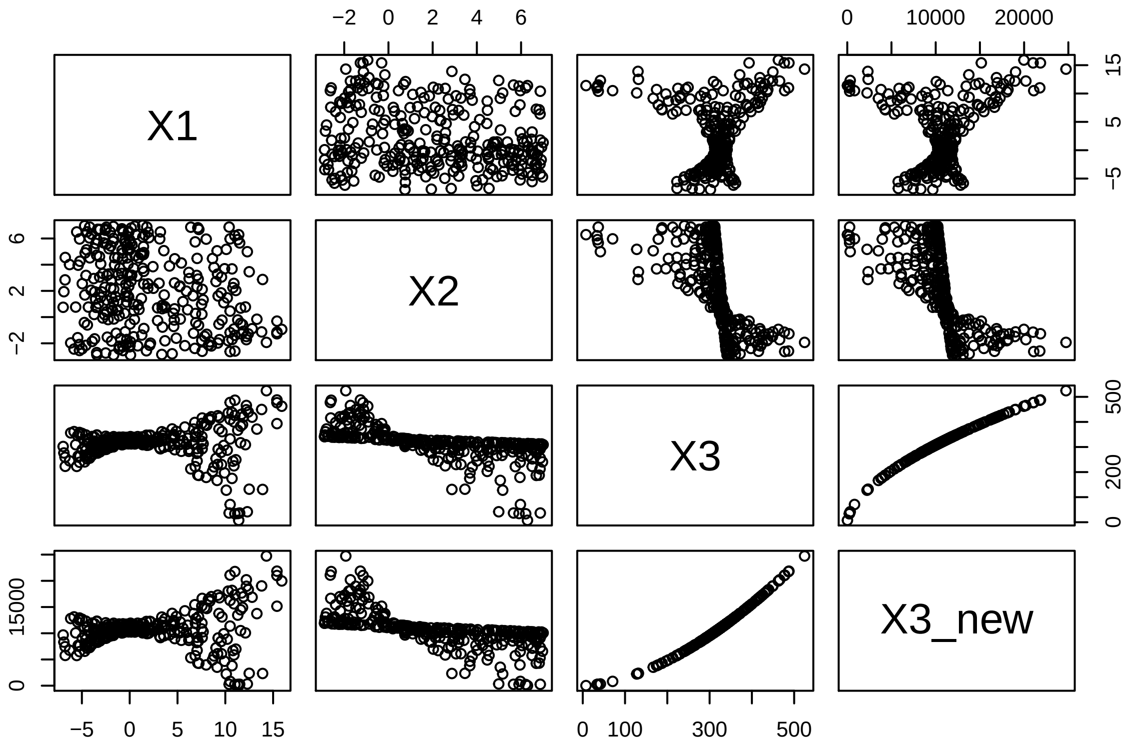
```
data_clean <- data[-c(201), ]
data_clean <- data_clean[-c(209, 75), ]
plot(data_clean)
```



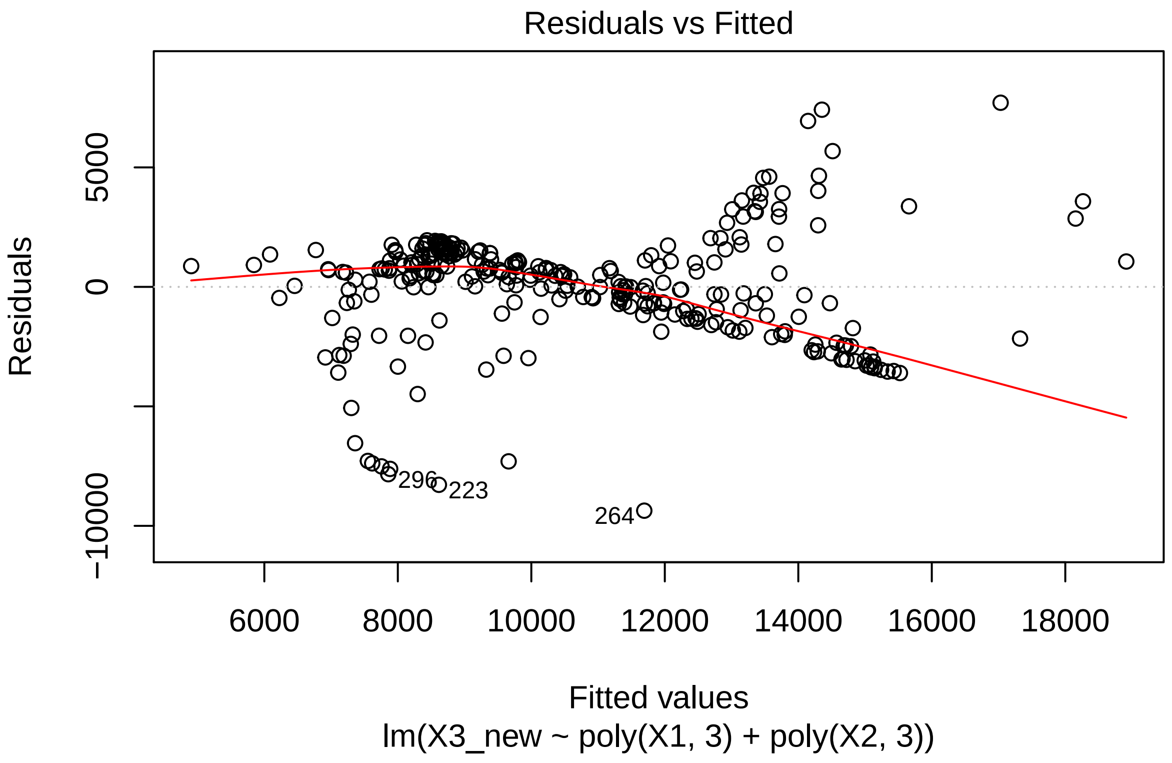
```
adj_factor <- abs(min(data_clean$X3)) + abs(mean(data_clean$X3))
data_clean$X3 <- data_clean$X3 + adj_factor
x3BC <- caret::BoxCoxTrans(data_clean$X3)
print(x3BC)
```

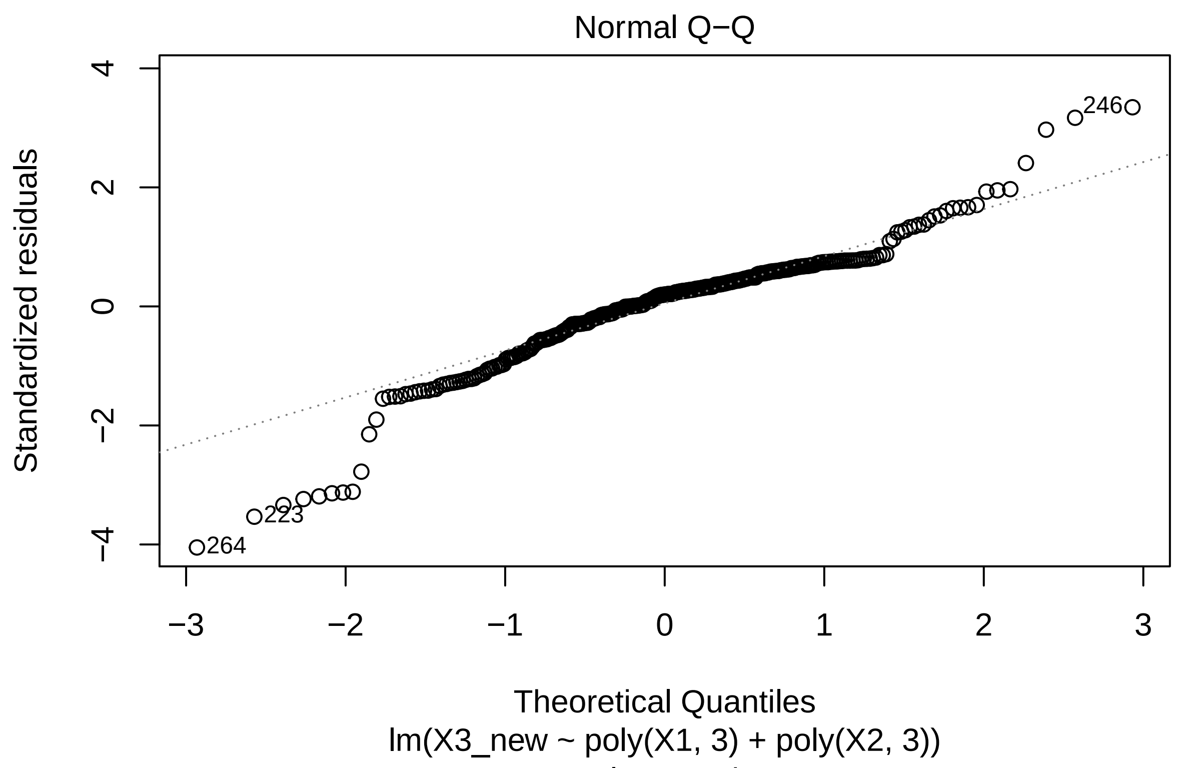
```
## Box-Cox Transformation
##
## 297 data points used to estimate Lambda
##
## Input data summary:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   7.665 299.665 319.725 315.000 338.965 524.665
##
## Largest/Smallest: 68.5
## Sample Skewness: -1.29
##
## Estimated Lambda: 1.7
```

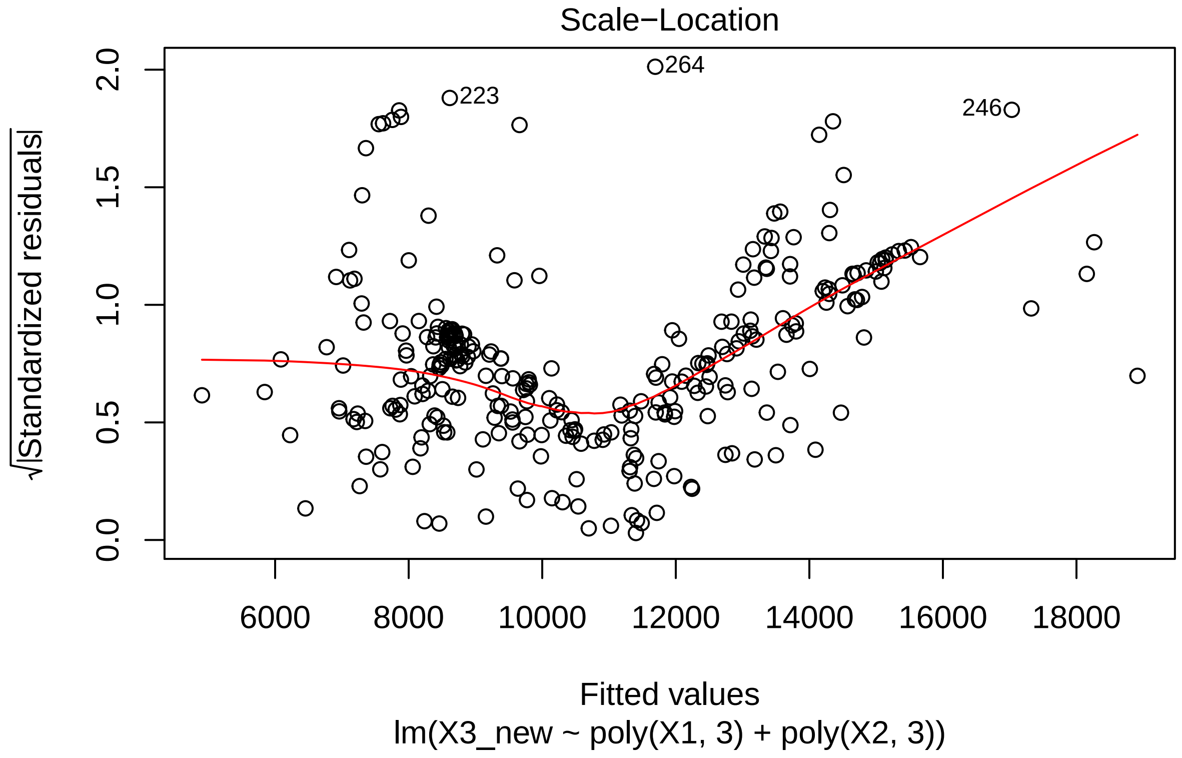
```
data_adj <- cbind(data_clean, X3_new=predict(x3BC, data_clean$X3))
plot(data_adj)
```

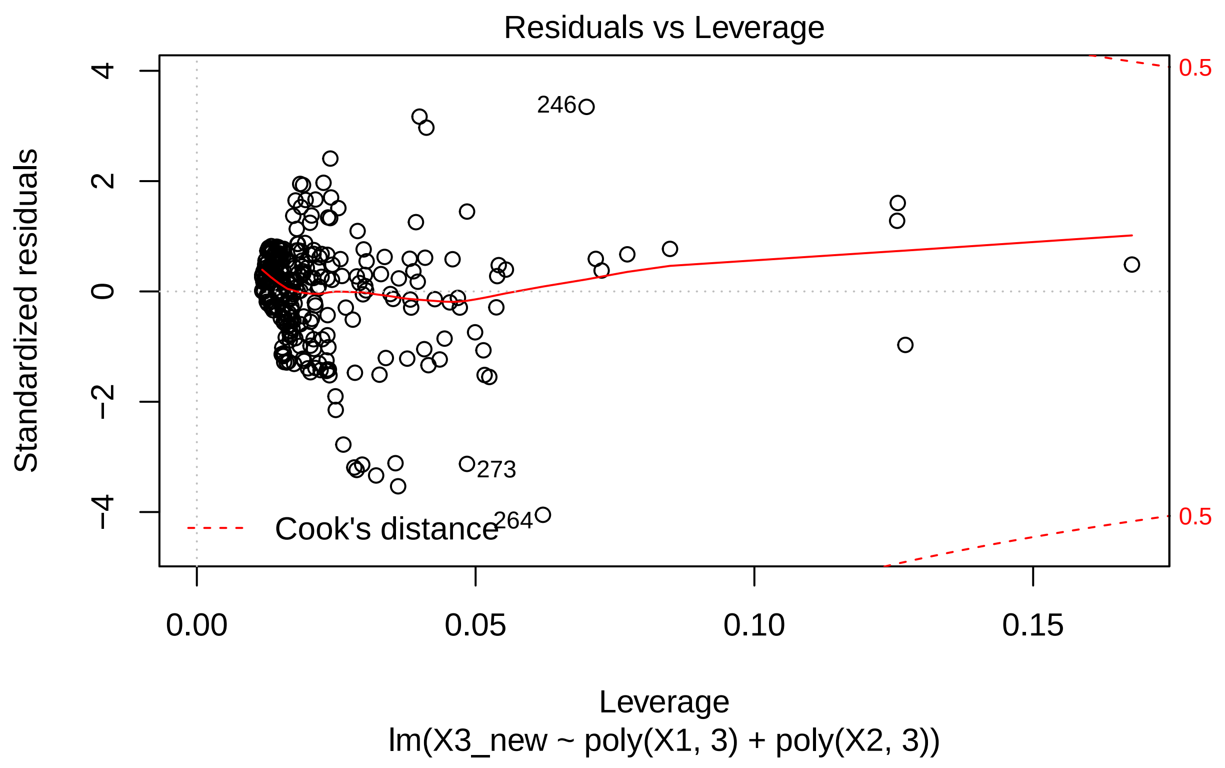


```r
reg_adj <- lm(X3_new ~ poly(X1, 3) +  poly(X2, 3) , data=data_adj)
plot(reg_adj)
```




Normal Q−Q
Standardized residuals
Theoretical Quantiles
lm(X3_new ~ poly(X1, 3) + poly(X2, 3))
246
223
264
−3
−2
−1
0
1
2
3
−4
−2
0
2
4


Scale−Location
√|Standardized residuals|
Fitted values
lm(X3_new ~ poly(X1, 3) + poly(X2, 3))
264
223
246
0.0
0.5
1.0
1.5
2.0
6000
8000
10000
12000
14000
16000
18000



```r
summary(reg_adj)
```

```
##
## Call:
## lm(formula = X3_new ~ poly(X1, 3) + poly(X2, 3), data = data_adj)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -9365.0 -1139.9   472.8  1360.2  7705.5
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)   10711.7      138.5  77.314  < 2e-16 ***
## poly(X1, 3)1   2888.0     2458.9   1.175  0.24115
## poly(X1, 3)2   -402.6     2404.1  -0.167  0.86713
## poly(X1, 3)3  17241.1     2395.0   7.199 5.23e-12 ***
## poly(X2, 3)1 -40667.4     2450.5 -16.595  < 2e-16 ***
## poly(X2, 3)2   7371.8     2392.1   3.082  0.00226 **
## poly(X2, 3)3   4419.1     2415.6   1.829  0.06836 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2388 on 290 degrees of freedom
## Multiple R-squared:  0.5546, Adjusted R-squared:  0.5454
## F-statistic: 60.18 on 6 and 290 DF,  p-value: < 2.2e-16
```

We first load the data and remove outlying data points through visual inspection. This is possible because there are just a few points, but in a larger data set, we could have to cull using conditionals. From viewing the dataset, we can clearly see a polynomial relationship. Without doing complex iterative regressions, we can estimate that this is a cubic regression in both X and Y. In addtion to this, we can see the scale of our response variable is orders of magnitude from out X and Y axes. To account for this, we can do a Box-Cox transformation to try and adjust for this.

Before we do that, we must adjust our response variable so it is above zero, and then apply the Box-Cox Lambda transform. After this we can create our polynomial regression and get our results. Here we can see that we obtain a decent R2 value of 0.5454, indicating that our regression is decent at predicting our data points after a transformation.