Outlier Detection Project

Sherman Aline & Ankita Bhattacharya

The project explores robust measures of estimation and multivariate methods of outlier detection. At the outset, let us explain why outliers are of interest in statistical analysis :

Outliers

Outliers are those data that deviate from global behavior of majority of data. Outliers or outlying observation have different definition in texts, for example "an outlier deviates so much from other observations as to arouse suspicions that it was generated by a different mechanism" (Hawkins)

Outliers can distort statistical inference in

- increasing error variance
- reducing the power of statistical tests
- causing biased estimates

This project focuses on detecting and analyzing outliers. The first part will deal with robustness measures through the sensibility curve for various location and scale estimators based on a numerical variable in our dataset. The second part illustrates two multivariate outlier detection methods - the Mahalanobis distance and Lof method through their implementation in R.

2.1 Dataset

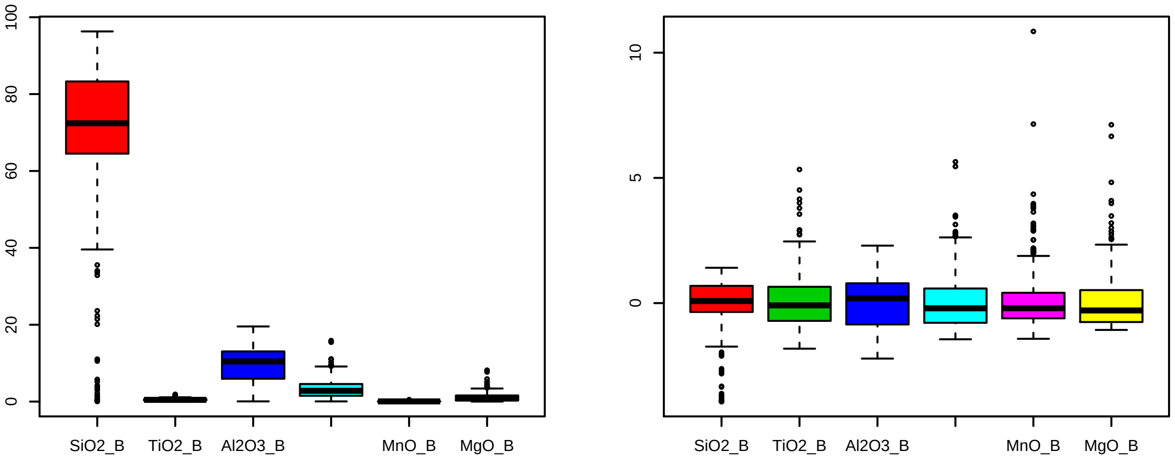
We choose a dataset available in the `mvoutlier`. This data contains 768 samples of the elements found in agricultural soils in Northern Europe.

We choose a subset of this data and narrow the study to 6 elements found in the soil.

```r
df <- bssbot[, c(5:10)]
```

First, we look at boxplots to see if our variables are similarly distributed. We compare boxplots of original (left) and scaled (right).

```r
df.std <- scale(df, center = T, scale = T); par(mfrow=c(1,2), cex=0.5)
boxplot(df, col = 2:7); boxplot(df.std, col = 2:7)
```



We see that the variable SiO2 has a very large spread and which might distort the reading of other variables. There are outliers for almost all variables except Al2O3. Al203 is left skewed while MgO is slightly right skewed. Otherwise the variables seem to be symmetric. The boxplot on standardized data shows the outliers are easier to read but we note that the standardization procedure is distorted by their presence.

2.2 Robustness measures: Sensitivity curve

2.2 1.) compute the value of the sensitivity curve at each of these three points for the mean, the median, the alpha-trimmed mean estimators, and the Huber M-estimator 1

We choose to build our sensitivity curve on the location estimators of our first variable, SiO2 because it shows the largest number of outliers.

The sensitivity curve measures the effect of a single observation (usually an outlier) on an estimator and was designed by John Tukey in 1970. The SC is defined as:

$SC_{n+1}(x^*) = (n+1)(T_{n+1} - T_n)$ where T_n is an estimator of the data in question e.g. mean, with n observations.

Our data set has 768 observations. For example, if we were to add the median and then check the sensitivity curve of the mean, we would simply find $(n+1)(\mu(x_1, ..., x_{768}, x_{median}) - \hat{\mu}(x_1, ..., x_{768}))$

```
# Naming this vector x
x <- df[, 1]
# A quick look at the location estimators
print(paste("mean:",mean(x),"med:",median(x),"trimmed mean:",mean(x, trim = 0.1),"huber:",huber(x)[1]))
```

```
## [1] "mean: 70.8835286458333 med: 72.41 trimmed mean: 73.2102597402597 huber: 73.0362271449566"
```

Three points were chosen as instructed: one below the minimum = -50, the median, one above the maximum = 150. We choose the limits to be -50 and 150 as it gives us a large spectrum centered around the median. Then we calculate the sensitivity curve for the following location estimators:

```
new.obs <- sort(c(seq(from= -50, to = 150, by= 10), median(df[, 1])))
sens.mean <- vector(length = length(new.obs))
sens.med  <- vector(length = length(new.obs))
sens.trimmed <- vector(length = length(new.obs))
sens.huber <- vector(length = length(new.obs))

for(i in (1:length(sens.mean))){
  sens.mean[i] =  (length(x)+1)*(mean(c(x, new.obs[[i]])) - mean(x))
  sens.med[i] = (length(x)+1)*(median(c(x, new.obs[[i]])) - median(x))
  sens.trimmed[i] = (length(x)+1)*(mean(c(x, new.obs[[i]]), trim = 0.1) - mean(x, trim = 0.1))
  sens.huber[i] = (length(x)+1)*(unlist(huber(c(x,new.obs[i]))[1]) - unlist(huber(x)[1]))
}
```

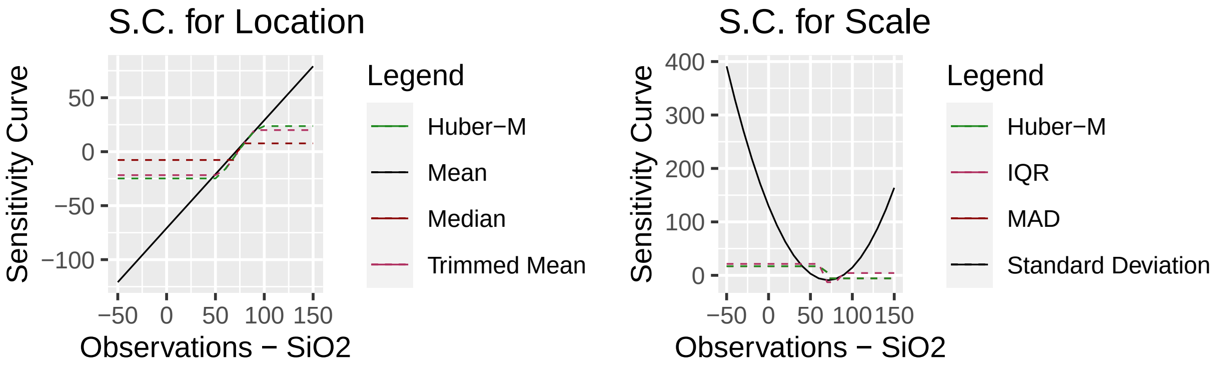
2.2 2.) compute the value of the sensitivity curve at each of these three points for the scale estimators SD, MAD, IQR and Huber M estimator

```
sens.sd <- vector(length = length(new.obs))
sens.mad  <- vector(length = length(new.obs))
sens.iqr <- vector(length = length(new.obs))
sens.huber.scale <- vector(length = length(new.obs))


for(i in (1:length(sens.sd))){
  sens.sd[i] =  (length(x)+1)*(sd(c(x, new.obs[[i]])) - sd(x))
  sens.mad[i] = (length(x)+1)*(mad(c(x, new.obs[[i]])) - mad(x))
  sens.iqr[i] = (length(x)+1)*(IQR(c(x, new.obs[[i]]))/1.349 - IQR(x)/1.349)
  sens.huber.scale[i] = (length(x)+1)*(unlist(huber(c(x,new.obs[i]))[2]) - unlist(huber(x)[2]))

}
```

2.2 3.) Plot the curves (see appendix for code)



2.2 4.) Which estimators are B-robust ?

When an estimator has a bounded influence function, it is said to be B-robust. For location estimators, the median, trimmed mean and Huber-M estimators are B-robust. For scale estimators MAD, IQR and the Huber-M estimators are B-robust

2.2 5.) What does it mean to have a sensitivity equals to 0 ?

Zero sensitivity implies that adding the particular observation to the sample leaves the estimator unchanged.

2.3 Outlier Detection

The Mahalanobis distance for a p-dimensional vector $X = (x_1, x_2, ..., x_p)^T$ with mean $\mu = (\mu_1, \mu_2, ..., \mu_p)^T$ and covariance matrix Σ is defined as $d_M(X, \mu) = \sqrt{(X-\mu)'\Sigma^{-1}(X-\mu)}$. The squared mahalanobis distance can be written as follows : $d_M^2(X, \mu) = (X-\mu)'\Sigma^{-1}(X-\mu)$

From the course, we have seen that for a given k, the Mahalanobis rule considers a multivariate observation to be an outlier if $(X-\mu)'\Sigma(X-\mu) \geq k^2$

In other words, we are to scale the contribution of individual variables to the distance value according to the variability of each variable. It differs from Euclidean distance in that it takes into account the correlations between variables in the data set. Clearly if $\Sigma_p = I_p$, the MD reduces to the Euclidean distance.

If we assume X to follow a multivariate normal distribution, $X \sim N_p(\mu, \Sigma_p)$, we can say that the squared distance follows a χ^2 distribution with p degrees of freedom.

We will see applications of this method, and the robust version of this measure and then answer the chief questions of interest.

Affine Invariance: The squared Mahalanobis distance is affine invariant as $d^2(AX + b) = d^2(X)$ for any full rank p x p matrix **A** and any p-vector **b**. For this we take the first 10 observations from the dataset and apply an affine transformation. We see that the mahalanobis distance for the original data and the transformed one is unchanged. This holds for both robust and non-robust methods.

```
# Affine transformation
df.affine <- 10*df[1:10, ] + 1
MD.affine <- mahalanobis(df.affine, colMeans(df.affine), var(df.affine))
MD.org <- mahalanobis(df[1:10, ], colMeans(df[1:10, ]), var(df[1:10, ]))
all.equal(MD.affine, MD.org)
```

```
## [1] TRUE
```

A note on the robust squared Mahalanobis distance Without robust estimators, we risk masking and swamping effects. The masking effect is where an outlier is declared only in the presence of another outlier, and would not have been otherwise. The swamping effect is where non-outlying data are mislabeled as outliers. A robust estimator should be immune to these.

We use Reweighted Minimum Covariance Determinant as our robust estimator. Due to space constraints, we do not cite the definition as seen in class, but we attempt to rephrase the idea. MCD calculates the mean and covariance matrix based on the most central subset of the data. The re-weighting step is applied to improve efficiency at normal data. Thus, MCD finds those h observations whose classical covariance matrix has the lowest possible determinant. The MCD estimate of location is the mean of h observations and the estimate of scatter is a sample covariance matrix of these h points (multiplied by consistency factor).

Distinguishability of outliers from non-outlier observations: In order to illustrate this point, we will compute the outliers in our dataset using both the ordinary and robust squared Mahalanobis distances.

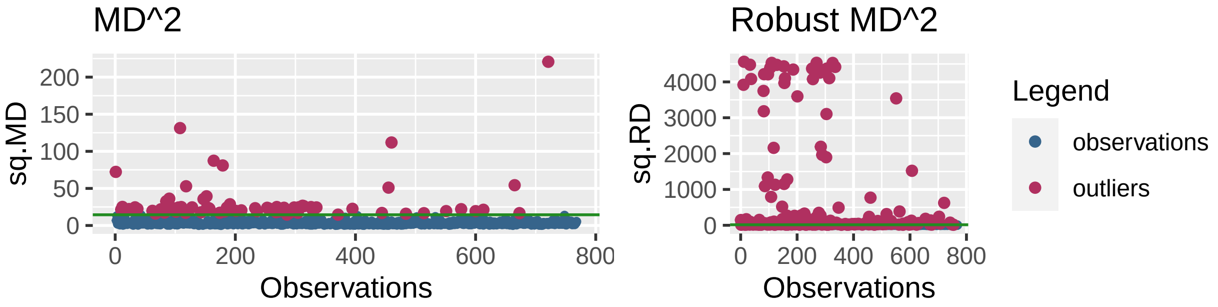
```
# Computing the squared mahalanobis distance(MD) - non-robust
MD <- mahalanobis(df, colMeans(df), var(df))
# Finding the cut-off based on the chi-sq quantile = 0.975 (5%)
cutoff.chi.sq <- qchisq(0.975, df = ncol(df))

# Robust squared MD :  Computing the MCD estimator with the size of the
#subsets over which the det is minimized = 0.75
RMCD <- CovMcd(df, alpha = 0.75)
# Compute the Mahalanobis distance with RMCD estimates
RD <- mahalanobis(df, RMCD@center, RMCD@cov)

# Identify which observations are outlying at level 5%.
outliers.RD <- which(RD > cutoff.chi.sq)
```

Let us now see how the chi-sq cutoff varies across robust and non-robust MD in classifying outliers. (See Appendix for code). Using the RMCD to define the robust mean and cov. matrix, we find **213** outliers vs. **69** in the non-robust mthod. This is a large difference!

```
colours <- c("observations" ="steelblue4", "outliers" = "maroon" )
p1 = ggplot() +
  geom_point(aes(y = as.numeric(MD), x = 1:length(MD), col = "observations"),  size = 1) +
  geom_point(aes(y = MD[which(MD > cutoff.chi.sq)], x = as.numeric(names(MD[which(MD > cutoff.chi.sq)]))
  geom_hline(yintercept = cutoff.chi.sq, col = "forestgreen" )+
  labs(x = "Observations", y = "sq.MD", color = "Legend")+
  scale_color_manual(values = colours) + ggtitle("MD^2") + theme_gray()+
  theme(legend.position="none")
p2 = ggplot() +
  geom_point(aes(y = as.numeric(RD), x = 1:length(RD), col = "observations"),  size = 1) +
  geom_point(aes(y = RD[outliers.RD], x = as.numeric(names(RD[outliers.RD])), col = "outliers")) +
  geom_hline(yintercept = cutoff.chi.sq, col = "forestgreen" )+
  labs(x = "Observations", y = "sq.RD", color = "Legend")+
  scale_color_manual(values = colours) +
  ggtitle("Robust MD^2") +
  theme_gray()
grid.arrange(p1, p2, ncol=2)
```

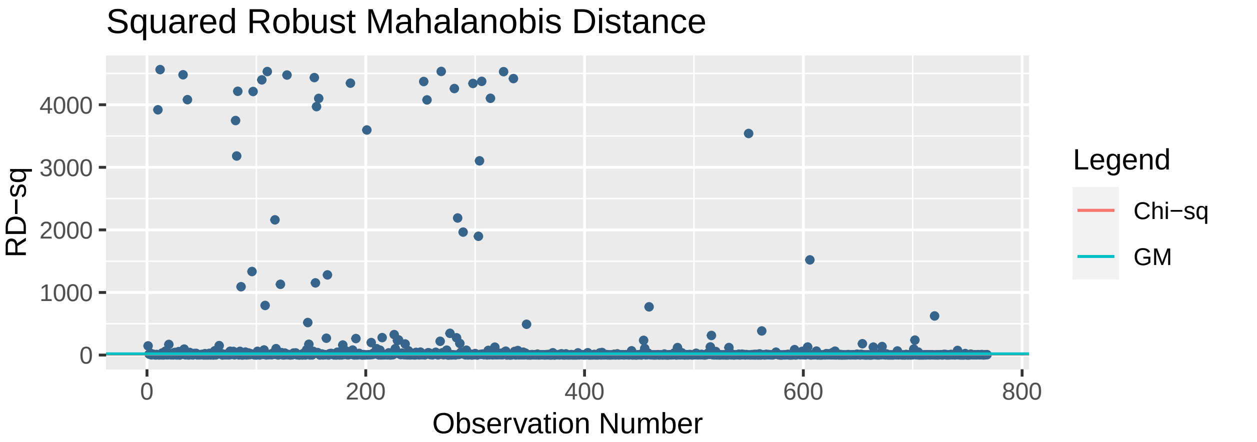


We try two robust methods : the Green & Martin method and the adjusted quantile plot.

```
# 1. Cut-off based Green and Martin (2014): CerioliOutlierDetection
cutoff.GM <- hr05CutoffMvnormal(n.obs = nrow(df), p.dim = ncol(df),
mcd.alpha = 0.75, signif.alpha = 0.025,method = "GM14",use.consistency.correction =T
)$cutoff.asy; outliers.RD.GM <- which(RD > cutoff.GM)
```

For all cut-offs we have α at the 5% level. The cut-off using method 1. is 20.37, which is not too large compared to the chi-square method, 14.45. The number of outliers detected by the former is 184, which is less than the robust MCD method based on the chi-sq distribution. Given the scale of the y-axis, both lines cannot be distinguished on the graph.

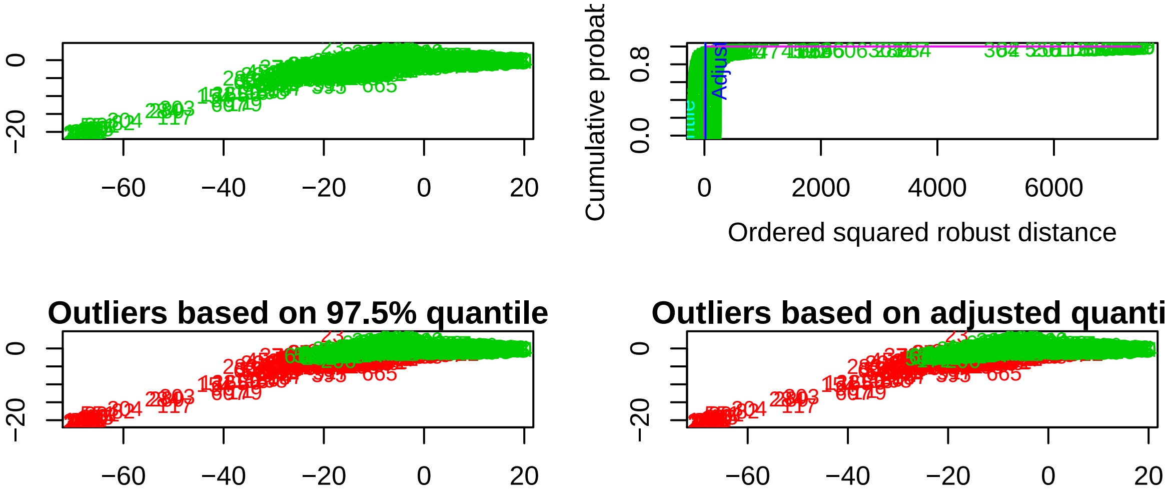
```
ggplot() +
  geom_point(aes(y = as.numeric(RD), x = 1:length(RD)), col = "steelblue4",  size = 1) +
  geom_hline(aes(yintercept = cutoff.chi.sq, col = "Chi-sq" ))+
  geom_hline(aes(yintercept = cutoff.GM, col = "GM" ))+
   labs(x = "Observation Number", y = "RD-sq", color = "Legend")+
  ggtitle("Squared Robust Mahalanobis Distance") + theme_gray()
```



2. Adjusted quantile plot

We use an adaptive procedure searching for outliers specifically in the tails of the distribution of the observations, also beginning at a certain chi-sq quantile, simply because outliers have **abnormally large** deviations in the tails. The cutoff based on the 97.5% quantile is smaller than the adjusted quantile. 257 outliers are declared by the latter method.

```
par(cex=0.5); aq.plot(df, alpha=0.75)
```



In conclusion, It is always better to look at a QQ-plot to find outlier as it requires no distributional assumptions. However, automatic outlier detection is better for large amounts of data.

Computational ease: This method is intuitive and easy to implement. Although the calculation becomes heavy in higher dimentions, many fast MCD calculations are proposed by R. We have listed the time taken to run each of the two processes.

```
# Time taken to calculate the squared MD
system.time(mahalanobis(df, colMeans(df), var(df)))
```

```
##    user  system elapsed 
##   0.000   0.000   0.001
```

```
# Time taken for robust calculation
system.time(c(RMCD <- CovMcd(df, alpha = 0.80),mahalanobis(df, RMCD@center, RMCD@cov)))
```

```
##    user  system elapsed 
##   0.112   0.002   0.134
```

Are the methods explainable ?: Explainability of a method refers to its ability in pointing out the variables for which the observation is outlying. This helps us have an explanation as to why an observation is outlying. For the Mahalanobis distance which takes into account all variables at once to produce a single measure, we do not know for which variables the an observation is identified as outlying.

LOF

Method Overview

Local Outlier Factor (LOF), relies on K-Nearest Neighbors to calculate distance.

For each observation o, a ball is drawn so that the K nearest observations reside in the ball. Two distances are considered from here:

- k-distance of an object o, distk(o): distance between o and its k-th nearest neighbor
- k-distance neighborhoodof o, $N_k(o) = [o'|o' \in D, dist(o,o') \leq dist_k(o)]$

And for two points o and o’,

reachdist(o,o’) = $max\{dist_k(o), dist(o,o')\}$

And local reachability distance is defined as:

$lrd_k(o) = \frac{||N_k(o)||}{\sum_{o' \in N_k(o)} reachdist(o,o')}$

Finally, the LOF for observation o is calculated as an average of the lrd of o, and the lrd of each of the k nearest neighbors.

$$LOF_k(o) = \frac{\sum_{o' \in N_k(o)} \frac{lrd_k(o')}{lrd_k(o)}}{||N_k(o)||}$$

So $LOF_k(o)$ is high when o is less dense relative to it's nearest neighbors.

In effect, this measure captures local outliers, as the measure is only considering kˆk observations, a subset of the entire datastet. Due to the numerical complexity of this method, it is not very fast.

Note that k is defined by the user, we will discuss it's optimal choice in the application.

We interpret the LOF scores to determine if an observation is outlying or not as follows:

- LOF ~ 1 : local density is not different from neighbors
- LOF < 1 : in a dense region of obsrvations, not outlying
- LOF > 1: observation is isolated, is an outlier.

Interpreting whether an observation is outlying or not is not clear-cut. Our calculations only consider their neighborhood in calculations, and as we will see this means determining the correct size k of the neighnborhood is vital to accurate outlier detection.

Without proper methods for determining the optimal k-size for a certain context, the LOF method proves to be less ideal than non-local approaches.

Application of Method and Comments

We use the package Rlof's implementation of LOF, as it allows faster computation of multiple values of k thorugh parallelization.

Since **LOF is not affine invariant**, we must standardize the data:

```
df <- as.data.frame(bssbot[,c(5:10)]) #reload original data
df <- as.data.frame(scale(df, scale=TRUE, center=TRUE))
```

In the original paper on LOF, the authors define some metrics for choosing k, which they refer to as MinPts.((Breunig) page 8)

- MinPtsLowerBound > 10, because in K<10 there is likely to be large variance in LOF scores which lead to inaccurate results. This was determined using Monte-Carlo experiments on uniformly distributed data.

- MinPtsUpperBound should be chosen such that observations from another one cluster don't get counted as outliers in a cluster.

We choose [11:55]. For the Upper Bound we are not sure what is a good choice, but we will comment more on this choice later.

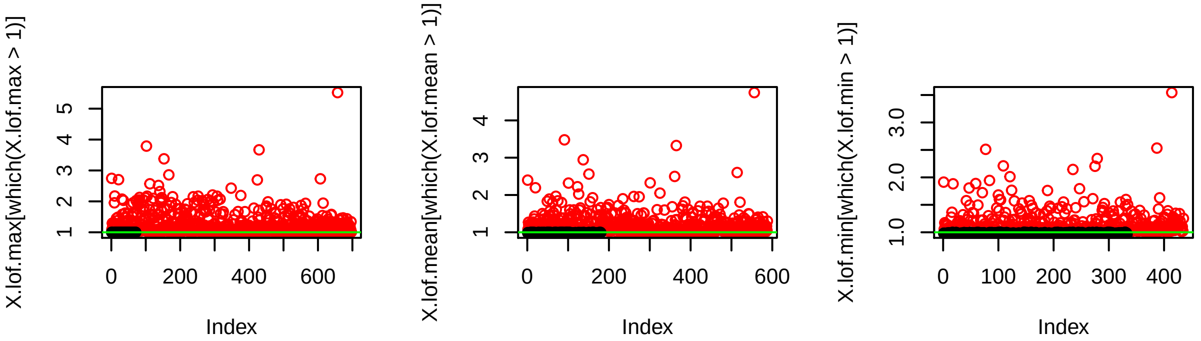
```
time.lof55 <- system.time(X.lof <- lof(df,11:55, cores = 2))
```

In the LOF paper, for aggregation it is suggested to take the maximum LOF from your range of calculations, because it highlights the instances when an observation is outlying. They mention that mean and minimum could be used but in most applications are likely to dilute the outlying-ness by hiding or making the outlying effect smaller.

We will try the mean and min aggregation approaches in order to compare. If they seem to perform better than maximum, then it seems likely that our choice of MinPtsUpperBound is too large, and overestimating outliers by including other clusters at outliers.

```
    X.lof.max <- apply(X.lof, 1, max); X.lof.mean <- apply(X.lof, 1, mean)
X.lof.min <- apply(X.lof, 1, min)
```

```
par(mfrow=c(1,3))
plot(X.lof.max[which(X.lof.max > 1)], col="red")
points(X.lof.max[which(X.lof.max <= 1)]); abline(h=1, col="green")
plot(X.lof.mean[which(X.lof.mean > 1)], col="red")
points(X.lof.mean[which(X.lof.mean <= 1)]); abline(h=1, col="green")
plot(X.lof.min[which(X.lof.min > 1)], col="red")
points(X.lof.min[which(X.lof.min <= 1)]); abline(h=1, col="green")
```



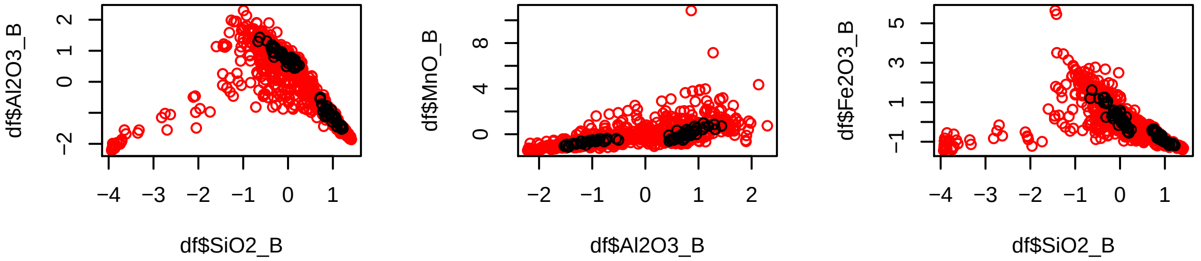
```
## [1] "max outliers:  697 max non-outliers:  71 max variance in LOF:  0.14864428765828"

## [1] "mean outliers:  588 mean non-outliers:  180 mean variance in LOF:  0.0897983008859361"

## [1] "min outliers:  435 min non-outliers:  333 min variance in LOF:  0.0459134486473675"
```

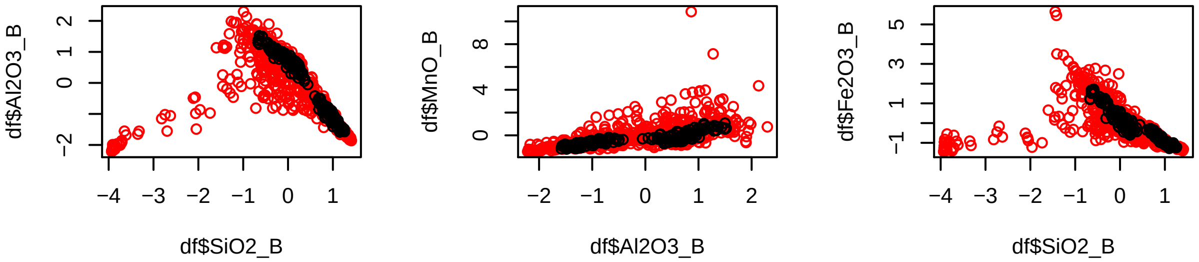
We can see from the graphs that there are much more outliers for the max, as expected, and that the variance in LOF score is higher. **So when max aggregation is used as directed, this method is easier to distinguish outliers as observations are further from 1.** The maximum also reports a majority of outliers, and only ∼70 observations which aren't outliers. We investigate further by plotting the projections of the data:

```
#max
par(mfrow=c(1,3));colored_outliers(df$SiO2_B,df$Al2O3_B, X.lof.max )
colored_outliers(df$Al2O3_B, df$MnO_B, X.lof.max )
colored_outliers(df$SiO2_B, df$Fe2O3_B, X.lof.max )
```

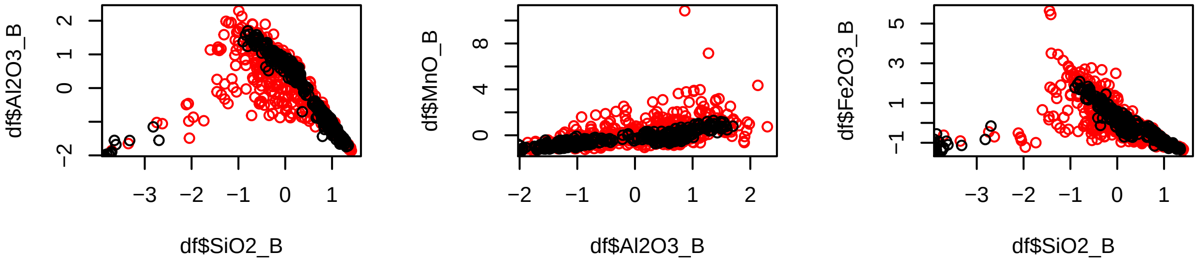


With the max aggregation, we seem a bit versensitive to outliers. We are left with two small clusters of around 30 points. This is despite what appears to be a common linear trend in our projections.

```
#mean
par(mfrow=c(1,3)); colored_outliers(df$SiO2_B,df$Al2O3_B, X.lof.mean )
colored_outliers(df$Al2O3_B, df$MnO_B, X.lof.mean )
colored_outliers(df$SiO2_B, df$Fe2O3_B, X.lof.mean )
```



```
#min
par(mfrow=c(1,3)); colored_outliers(df$SiO2_B,df$Al2O3_B, X.lof.min )
colored_outliers(df$Al2O3_B, df$MnO_B, X.lof.min ); colored_outliers(df$SiO2_B, df$Fe2O3_B, X.lof.min )
```

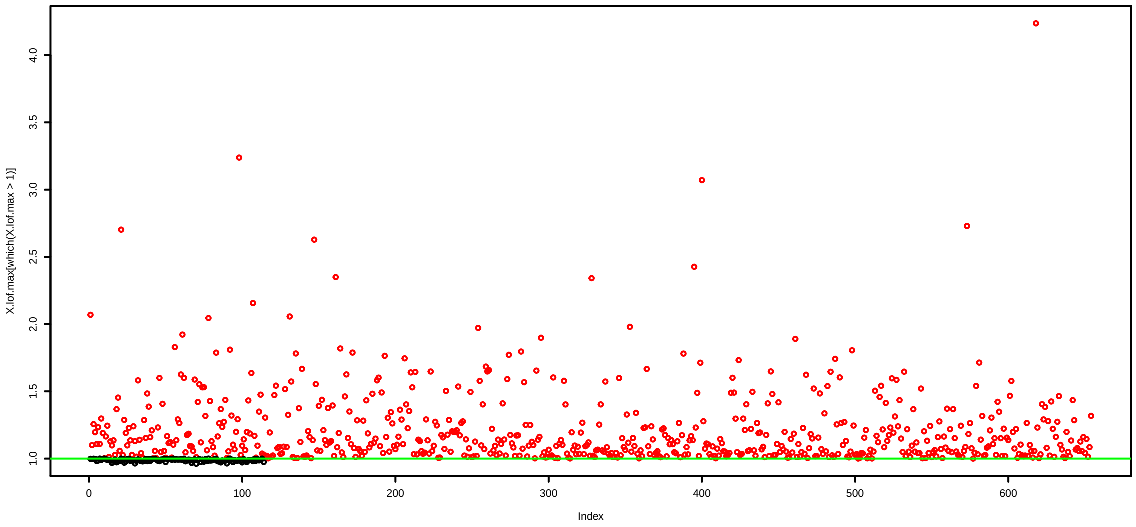


Using the mean, we have a higher number of non-outliers and there appears to be more contingulty between our two clusters. In the minimum method, we observe a small third cluster forming, and much more spread forming in our data.

Overall, we believe mean performs best for our data. The clusters are not unreasonablty small, and they are still dense and visually discernable as contiguous areas. However it is still possible these two cluster would be better interpreted as one. The minimum method is not ideal as it starts to include points which are in very non-dense areas. In particular we can see in the third graph that some points in very low-density area between the cluster and the line are included.

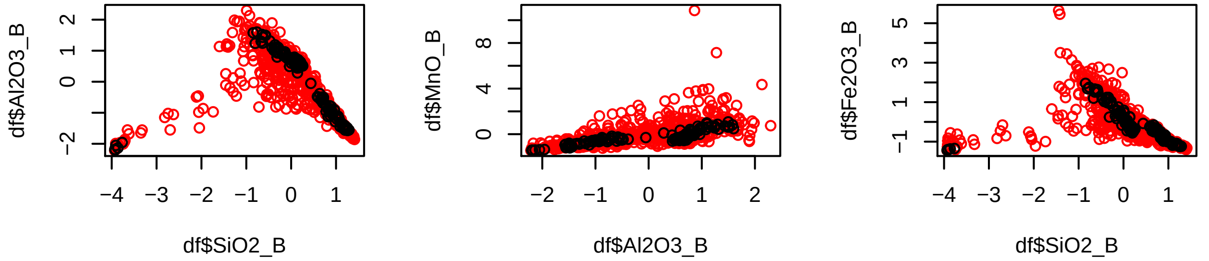
Since mean seems to perform bettwe we suspect our upper bound for K is misspecified. We repeat with maximum, and a smaller MinPtsUpperBound:

```
time.lof20 <- system.time(X.lof <- lof(df,11:20, cores = 2)); X.lof.max <- apply(X.lof, 1, max)
par(cex=0.3); plot(X.lof.max[which(X.lof.max > 1)], col="red")
points(X.lof.max[which(X.lof.max <= 1)]); abline(h=1, col="green")
```



```
## [1] "max outliers:  654 max non-outliers:  114 max variance in LOF:  0.081943181309679"
```

```r
par(mfrow=c(1,3))
colored_outliers(df$SiO2_B,df$Al2O3_B, X.lof.max ); colored_outliers(df$Al2O3_B, df$MnO_B, X.lof.max )
colored_outliers(df$SiO2_B, df$Fe2O3_B, X.lof.max )
```



This seems reasonably improved. It is hard to interpret whether the small cluster in the bottom right should be considered as a cluster or all outliers, but more knowledge of the nature of geological data may give practical users more insight into this.

There appear to be roughly 3 clusters now, and they are better separated than with the mean and min aggregation methods.

Our results, and the difficulty in interpreting the, reflect the **lack of explainability in this method**. If the data is more clearly separated into contiguous groups, then perhaps the method is slightly more explainable.

Speed: Finally, let's compare the times of our methods.

```r
time.lof20; time.lof55
```

```
##    user  system elapsed
##   1.430   0.446   2.318

##    user  system elapsed
##   5.504   0.914  11.346
```

As expected this method is significantly slower than MD, due to the numerical complexity and the recommendation of many values of k. For fewer values, times improve significantly.

Appendix & Citations

Sensitivity curve code:

```r
# Plotting the location estimators
colors <- c("Mean" = "black", "Median" = "darkred", "Trimmed Mean" = "maroon", "Huber-M" = "forestgreen
colors1 <- c("Standard Deviation" = "black", "MAD" = "darkred", "IQR" = "maroon",
            "Huber-M" = "forestgreen")

pltone <- ggplot() + # Plotting the sensitivity curve of the four location estimators.
  geom_line(aes(x = new.obs, y = sens.mean, col = "Mean"), lty = 1, lwd = 0.3 )+
  geom_line(aes(x = new.obs, y = sens.med, col = "Median"), lty = 2, lwd = 0.3 )+
  geom_line(aes(x = new.obs, y = sens.trimmed, col = "Trimmed Mean"), lty = 2, lwd = 0.3 )+
  geom_line(aes(x = new.obs, y = sens.huber, col = "Huber-M"), lty = 2, lwd = 0.3 )+
  labs(x = "Observations - SiO2", y = "Sensitivity Curve", color = "Legend",  title = "S.C. for Locatio
  scale_color_manual(values = colors)+ theme_gray()
# Plotting the scale estimators.
plttwo <- ggplot() +
  geom_line(aes(x = new.obs, y = sens.sd, col = "Standard Deviation"), lty = 1, lwd = 0.3 )+
  geom_line(aes(x = new.obs, y = sens.mad, col = "MAD"), lty = 2, lwd = 0.3 )+
  geom_line(aes(x = new.obs, y = sens.iqr, col = "IQR"), lty = 2, lwd = 0.3 )+
  geom_line(aes(x = new.obs, y = sens.huber.scale, col = "Huber-M"), lty = 2, lwd = 0.3 )+
  labs(x = "Observations - SiO2", y = "Sensitivity Curve",
       title = "S.C. for Scale", color = "Legend")+
  scale_color_manual(values = colors1)

grid.arrange(pltone,plttwo,nrow=1,ncol=2)
```

Outlier drawing function:

```r
#outlier printing function
colored_outliers <- function(x, y, mylist){
  #titles
  plot(x[which(mylist > 1)], y[which(mylist > 1)], col="red",
       ylab=deparse(substitute(y)), xlab=deparse(substitute(x)))
points(x[which(mylist <= 1)], y[which(mylist <= 1)])
}
```

Breunig, Markus. "LOF: Identifying Density-Based Local Outliers." https://www.dbs.ifi.lmu.de/Publikationen/Papers/LOF.pdf.

Hawkins, D. "Identification of Outliers." https://doi.org/10.1007/978-94-015-3994-4.